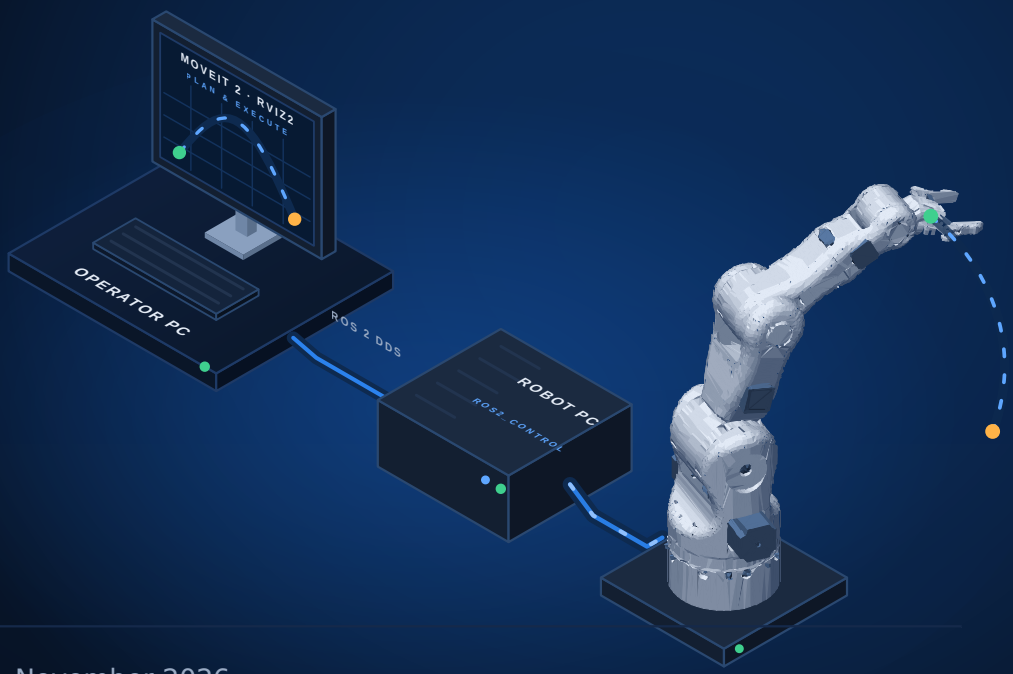
SmartManipulator-X · Operator PC

SmartManipulator-X

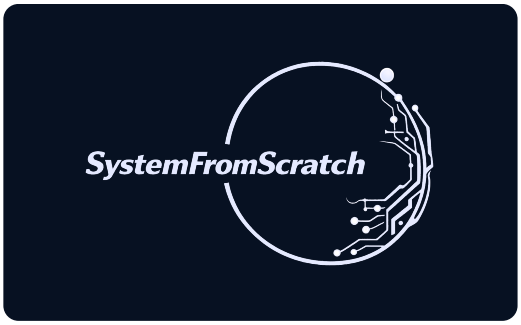
Operator PC: MoveIt 2 and RViz2 in Docker

USER MANUAL



Version dev-34d818b · November 2026
Anton Vynohradov
systemfromscratch.com

Imprint



SmartManipulator-X Operator PC User Manual
Version dev-34d818b
Edition: November 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
ORCID: orcid.org/0009-0005-6965-0911
E-mail: avynohradov@systemfromscratch.com

Online
Documentation: https://systemfromscratch.github.io/sfs-smartmanipulator-x-operator/
Source code: https://github.com/SystemFromScratch/sfs-smartmanipulator-x-operator

Copyright © 2026 Anton Vynohradov.

The project-owned files of SmartManipulator-X and this manual are licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in Appendix A. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SystemFromScratch® and the SystemFromScratch® logo are registered trademarks. The Apache License 2.0 grants no rights to use them; they may not be used to endorse or promote products derived from SmartManipulator-X without prior written permission.

ROS is a trademark of Open Source Robotics Foundation, Inc. MoveIt is a trademark of PickNik Inc. ROBOTIS, DYNAMIXEL and OpenMANIPULATOR are trademarks of ROBOTIS Co., Ltd. SmartFusion2 is a trademark of Microchip Technology Inc. BCN3D is a trademark of BCN3D Technologies. Docker is a trademark of Docker, Inc. Windows is a trademark of Microsoft Corporation. SmartManipulator-X is not affiliated with or endorsed by these companies. All other trademarks are the property of their respective owners.

This manual is generated from the same sources as the online documentation. The information in it was checked against the version stated above; later versions may differ.

Contents

1 Overview 5
- 1.1 What the Operator PC does 5
- 1.2 System components 6
- 1.3 Repositories 7
- 1.4 Joint contract 7
- 1.5 Current status 8

2 Getting started 9
- 2.1 Prerequisites 9
- 2.2 Clone both repositories side by side 9
- 2.3 1. Start the Robot PC 9
- 2.4 2. Start the Operator PC 10
- 2.5 3. Plan and execute 10
- 2.6 Shell on the same network 10
- 2.7 Stop and rebuild 11
- 2.8 Next steps 11

3 Planning in RViz2 12
- 3.1 Before you start 12
- 3.2 1. The window 13
- 3.3 2. Choose a named goal 14
- 3.4 3. Plan and check the preview 15
- 3.5 4. Execute 16
- 3.6 5. Set a goal with the interactive marker 17
- 3.7 6. Joints tab 18
- 3.8 7. The gripper 19
- 3.9 8. Speed of the motion 20
- 3.10 9. Planner options 21
- 3.11 What happens on Execute 22
- 3.12 Saving the layout 22
- 3.13 move_group without RViz2 23

4 Development 24
- 4.1 Repository layout 24
- 4.2 Working on the MoveIt configuration 24
- 4.3 VS Code tasks 25
- 4.4 Documentation 25
- 4.5 CI 26

5 Architecture 27
- 5.1 ROS 2 graph 27
- 5.2 What crosses the network 28
- 5.3 The robot model on both sides 28
- 5.4 Responsibilities and boundaries 29
- 5.5 Real robot and simulation 29
- 5.6 Package layout 29

6 MoveIt 2 configuration 31

6.1 Semantic description (SRDF) ... 31
6.2 Kinematics ... 32
6.3 Joint limits ... 33
6.4 Controllers ... 33
6.5 Launch file ... 34
6.6 Changing the configuration ... 34

7 Docker image and Compose ... 35
7.1 The image ... 35
7.2 Build contexts ... 36
7.3 Services ... 37
7.4 Environment ... 37
7.5 GUI on Windows 11 (WSLg) ... 38
7.6 GUI on a Linux desktop (X11) ... 38
7.7 Rendering ... 38

8 Network and DDS ... 39
8.1 One computer: smx_net ... 39
8.2 The Fast DDS profile: UDP only ... 39
8.3 Two computers: real Ethernet ... 40
8.4 Native Windows ROS 2 cannot join ... 40
8.5 SSH is not a ROS transport ... 40
8.6 Checklist ... 40

9 Troubleshooting ... 42
9.1 Normal messages ... 42
9.2 Start and build ... 42
9.3 Display ... 43
9.4 Discovery and data ... 43
9.5 Planning and execution ... 43
9.6 Model mismatch ... 44

Author & Company ... 45
Author ... 45
SystemFromScratch® ... 46
Contact ... 46
Support and feedback ... 47
Trademarks and acknowledgements ... 47

Appendix A License ... 48
A.1 What the license allows ... 48
A.2 What it requires ... 48
A.3 What it does not give ... 48
A.4 Trademarks ... 49
A.5 Full license text ... 49

CHAPTER 1

Overview

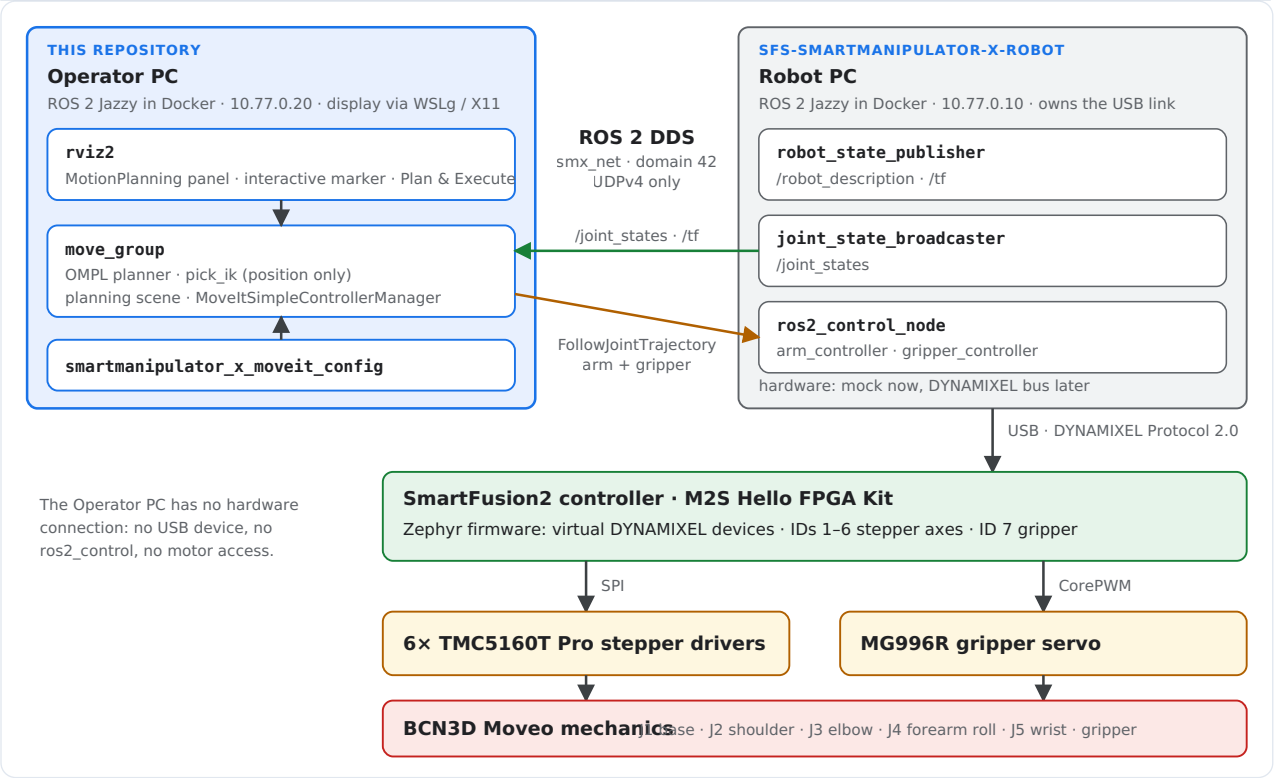
SmartManipulator-X is a robot arm project. It combines the ROS 2 control and operator-software concepts of ROBOTIS OpenMANIPULATOR-X (emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/) with the mechanical design of BCN3D Moveo (github.com/BCN3D/BCN3D-Moveo). A Microchip SmartFusion2 controller drives the stepper motors of the Moveo joints and a hobby servo on the gripper, and presents them to the ROS 2 host as virtual DYNAMIXEL devices over Protocol 2.0.

This documentation covers the **Operator PC**: the computer on which the operator plans motions and watches the arm. The whole system — the Robot PC, the controller firmware, the hardware and the mechanics — is documented on the SmartManipulator-X system site (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/), which is published from the Robot PC repository.

1.1 What the Operator PC does

Owns	Does not own
MoveIt 2 motion planning (`move_group`, OMPL, pick_ik)	The USB link to the SmartFusion2 controller (`/dev/ttyACM0`)
RViz2 with the MotionPlanning panel	`ros2_control`, the controllers and the hardware interface
The semantic robot description (SRDF), kinematics, planning limits, controller mapping	The robot description (URDF/Xacro and meshes) — it is consumed from the Robot PC repository
Its Docker image, Compose services and GUI setup	The motor drivers, the firmware, the power and the emergency stop
Simulation workflows (Gazebo: planned, not in this repository yet)	

The Operator PC never talks to the motor controllers. It plans in MoveIt and sends the trajectories as `FollowJointTrajectory` actions over ROS 2 DDS to the Robot PC, which runs `ros2_control`:



SmartManipulator-X with the Operator PC in focus: trajectories go over ROS 2 DDS to the Robot PC, which drives the controller and the arm

1.2 System components

Component	Role	Platform
Operator PC (this repository)	Motion planning and visualization: MoveIt 2, RViz2; simulation later	ROS 2 Jazzy in Docker; DDS client of the Robot PC
Robot PC	Robot-side ROS 2 control: `ros2_control`, the ROBOTIS hardware interface and DynamixelSDK; owns the USB link to the controller	Ubuntu 24.04, ROS 2 Jazzy in Docker
Controller firmware	DYNAMIXEL Protocol 2.0 device: six stepper axes (IDs 1–6), the gripper servo (ID 7) and a controller device (ID 200)	Zephyr RTOS on the SmartFusion2 Cortex-M3
FPGA design	Pin routing of SPI and UART, CorePWM for the gripper servo	SmartFusion2 FPGA fabric
Drives	Six TMC5160 modules in motion-controller mode, MG996R gripper servo	24 V and 6 V supplies
Mechanics	Joints, links and printed parts	BCN3D Moveo

1.3 Repositories

Repository	Content	Documentation
sfs-smartmanipulator-x-operator	**This repository**: Operator PC image, MoveIt 2 configuration, RViz2 setup, this documentation	This site; the system view in Operator PC (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/operator-pc/)
sfs-smartmanipulator-x-robot	Robot PC image and ROS 2 workspace, the robot description generated from the Moveo CAD, the Fast DDS profile, the system documentation	Robot PC (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/robot-pc/), BCN3D Moveo mechanics (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/mechanical/)
SmartManipulator-X-Controller	Zephyr firmware of the SmartFusion2 and the motor tester GUI	Firmware (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/firmware/), Motor tester GUI (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/gui/)
SmartManipulator-X-Libero	FPGA design of the SmartFusion2	Hardware (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/wiring/)

The two computer repositories are checked out **side by side**. The Operator PC image build reads the robot description and the DDS profile from the Robot PC checkout (default path `../SmartManipulator-X-Robot-PC`), see Docker image and Compose (p. 36).

1.4 Joint contract

The Operator PC, the Robot PC and the real hardware use the same joint names. They are defined in `smartmanipulator_x_description` in the Robot PC repository; the Operator PC only refers to them.

Joint	Moveo axis	DYNAMIXEL ID (SmartFusion2)	MoveIt group	Controller
`j1_base`	J1 base	1	`arm`	`arm_controller`
`j2_shoulder`	J2 shoulder	2 and 3 (two mirrored motors)	`arm`	`arm_controller`
`j3_elbow`	J3 elbow	4	`arm`	`arm_controller`
`j4_forearm_roll`	J4 forearm roll	5	`arm`	`arm_controller`
`j5_wrist`	J5 wrist	6	`arm`	`arm_controller`
`gripper`	gripper servo	7	`gripper`	`gripper_controller`

The gripper linkage has mimic joints (`gripper_idler_gear`, `gripper_pivot_*`, `gripper_finger_*`) that follow `gripper`; they are neither planned nor commanded.

1.5 Current status

- **Operator PC (this repository):** ROS 2 Jazzy image with MoveIt 2, pick_ik and RViz2; `move_group` and RViz2 start with one Compose command, join `smx_net`, plan with OMPL and execute through the Robot PC's `arm_controller` and `gripper_controller`, with the Robot PC running mock hardware on the same computer.
- **Robot model:** generated from the BCN3D Moveo CAD in the Robot PC repository. Joint limits are still placeholders until measured on the real arm; the MoveIt joint limits of this repository are conservative placeholders as well.
- **Real hardware:** not reachable from the Operator PC yet — the Robot PC refuses `use_mock_hardware:=false` until its real-hardware `ros2_control` path is verified (see the Robot PC documentation (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/robot-pc/)).
- **Open items:** Gazebo simulation on the Operator PC, measured joint limits and velocities, two-host operation over real Ethernet, orientation goals with a 5-DOF-aware IK setting.

CHAPTER 2

Getting started

This guide takes you from two fresh clones to RViz2 planning motions for the SmartManipulator-X arm. Everything runs in Docker on one computer: the Robot PC with mock hardware (every commanded joint position becomes the joint state at once) and the Operator PC on the same Docker network. No controller board and no arm are needed.

2.1 Prerequisites

Where	What
Windows 11	Docker Desktop with the WSL 2 backend. RViz2 is shown through WSLg, which Docker Desktop shares with its containers
Linux desktop	Docker Engine and an X11 session (Wayland desktops with XWayland work too)
Both	Git, Docker Compose **v2.17 or later** (for `additional_contexts`) and BuildKit (the default)
Optional	VS Code

ROS 2 is not installed on the host: it comes with the images (`ros:jazzy-ros-base`). Check the Compose version with `docker compose version`.

2.2 Clone both repositories side by side

The Operator PC image takes the robot description and the Fast DDS profile from the Robot PC repository. By default it looks for it next to this repository, under the name `SmartManipulator-X-Robot-PC`:

```
git clone https://github.com/SystemFromScratch/sfs-smartmanipulator-x-robot.git
SmartManipulator-X-Robot-PC
git clone https://github.com/SystemFromScratch/sfs-smartmanipulator-x-operator.git SmartMa
nipulator-X-Operator-PC
```

```
SmartManipulator-X/
├── SmartManipulator-X-Robot-PC/      Robot PC: ros2_control, robot description, DDS
profile
└── SmartManipulator-X-Operator-PC/   Operator PC: this repository
```

If the Robot PC checkout lives elsewhere, set `SMX_ROBOT_PC_REPO` in `docker/.env` (see Docker image and Compose (p. 37)).

2.3 1. Start the Robot PC

In the Robot PC repository:

```
docker compose -f docker/docker-compose.yml up --build robot_pc
```

The first run builds the image, which takes several minutes. Compose creates the Docker network `smx_net` (10.77.0.0/24), which stands in for the Ethernet link between the two computers, and starts the Robot PC at 10.77.0.10 with `robot_state_publisher`, `ros2_control_node` (mock hardware), `joint_state_broadcaster`, `arm_controller` and `gripper_controller`. Leave the terminal open.

> **Start the Robot PC first**
>
> The Operator PC declares `smx_net` as an **external** network. If the Robot PC has not created it yet, Compose stops with `network smx_net declared as external, but could not be found.`

2.4 2. Start the Operator PC

In this repository, in a second terminal:

```
docker compose -f docker/docker-compose.yml up --build operator_pc
```

The first build installs MoveIt 2, pick_ik and RViz2 on top of `ros:jazzy-ros-base` and builds the workspace (several minutes). The container joins `smx_net` at 10.77.0.20 and runs:

```
ros2 launch smartmanipulator_x_moveit_config operator.launch.py
```

which starts `move_group` and RViz2. The log of `move_group` ends with `You can start planning now!` once it has received the robot state from the Robot PC.

2.5 3. Plan and execute

RViz2 opens with the **MotionPlanning** display, planning group `arm`: the current state of the arm (as reported by the Robot PC) is opaque, the goal state is orange and semi-transparent.

1. Drag the interactive marker at the gripper (`tool_link`) to a new position.
2. In the **Planning** tab, click **Plan & Execute**.
3. The Robot PC log shows `arm_controller` accepting and finishing the goal. The arm in RViz2 follows the `/joint_states` that come back from the Robot PC.

For the named poses, the gripper and the planning options, see Planning in RViz2 (p. 12).

2.6 Shell on the same network

The `shell` service (profile `tools`) is a second container with the same image, network and GUI setup, for the ROS 2 command line:

```
docker compose -f docker/docker-compose.yml run --rm shell
```

```
ros2 node list                    # /move_group, /rviz2 and the Robot PC nodes
ros2 control list_controllers     # arm_controller, gripper_controller,
joint_state_broadcaster
ros2 topic echo /joint_states --once
glxgears                          # checks the display path
```

2.7 Stop and rebuild

Stop with **Ctrl+C**, or remove the Operator PC containers (the network belongs to the Robot PC project and stays):

```
docker compose -f docker/docker-compose.yml down
```

The workspace and the robot description are built **into** the image, not mounted. After changing the MoveIt configuration here, or the robot description in the Robot PC repository, start with `up --build` again.

2.8 Next steps

- Planning in RViz2 (p. 12): named poses, the gripper, planning options.
- Architecture (p. 27): which node runs where and what goes over DDS.
- Network and DDS (p. 39): two computers on a real network.

CHAPTER 3

Planning in RViz2

RViz2 is the operator interface of the Operator PC. It starts with `rviz/moveit.rviz` from `smartmanipulator_x_moveit_config` and talks to `move_group` through the MoveIt **MotionPlanning** plugin. Every plan is computed on the Operator PC; only the execution goes to the Robot PC.

This page walks through a complete session: start, a named goal, planning, execution, a goal set with the interactive marker, the joint view, the gripper and the planner settings. The screenshots were taken with the Robot PC running mock hardware on `smx_net`; with mock hardware every commanded position becomes the joint state at once, so the arm in RViz2 follows the trajectory exactly.

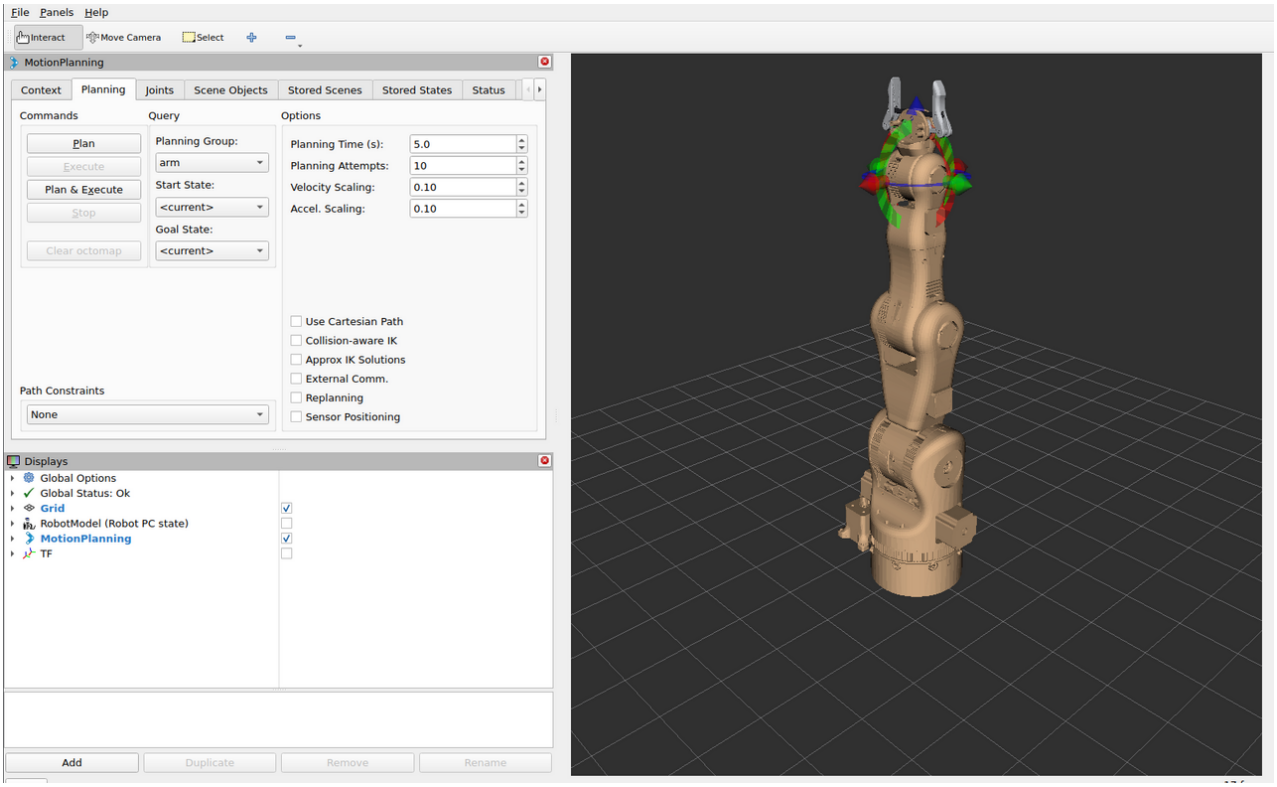
3.1 Before you start

Check	How
The Robot PC runs	In the Robot PC repository: `docker compose -f docker/docker-compose.yml up robot_pc`
The Operator PC runs	In this repository: `docker compose -f docker/docker-compose.yml up operator_pc` (Getting started)
`move_group` is ready	Its log ends with `You can start planning now!`

Two `ERROR` lines in the log at start-up are normal and can be ignored:

Message	Meaning
`No 3D sensor plugin(s) defined for octomap updates`	No depth camera is configured; the planning scene has no octomap
`Action server: /recognize_objects not available`	The *Detect* button of the Scene Objects tab has no object recognition behind it

3.2 1. The window



RViz2 after start: the MotionPlanning panel on the left with the Planning tab, the Displays list below it, and the 3D view on the right with the arm in its current state and the interactive marker at the gripper

Area	Content
Toolbar	**Interact** (move the marker, the default tool), **Move Camera**, **Select**
MotionPlanning panel	Tabs *Context*, *Planning*, *Joints*, *Scene Objects*, *Stored Scenes*, *Stored States*, *Status*, *Manipulation* (behind the arrow)
Displays	Grid, RobotModel (Robot PC state), MotionPlanning, TF — tick a box to show a display
3D view	The arm, the goal and the planned path in the fixed frame `world`

What the 3D view shows:

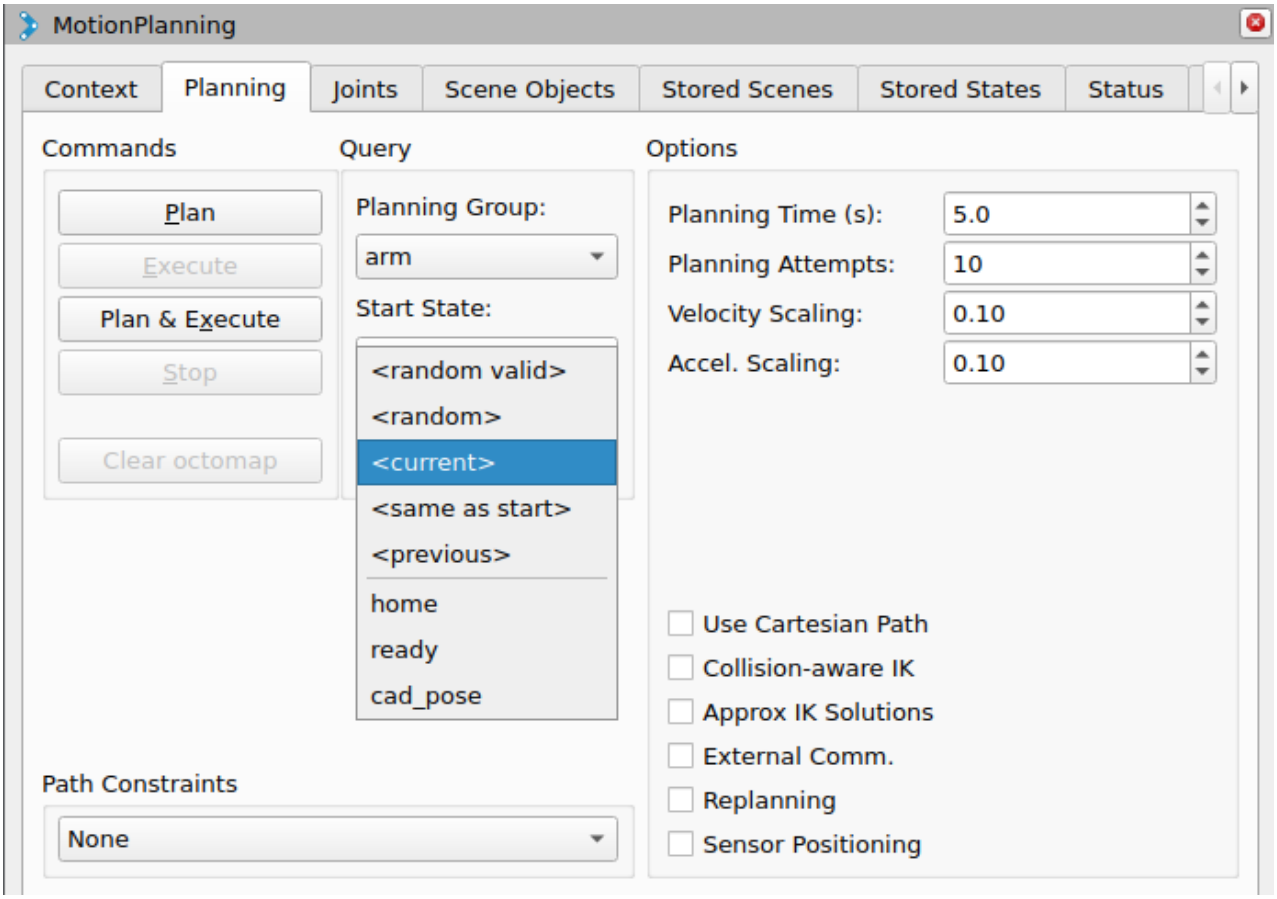
You see	It is
Arm in its own colours (light grey, metal parts)	The **current state**, as reported by the Robot PC in `/joint_states`
Orange, semi-transparent arm	The **goal state** (35 % opacity). Where it overlaps the current state the arm looks tan, as right after start
Red, green and blue arrows and rings at the gripper	The **interactive marker** on `tool_link`, the tip of the `arm` chain
Grey arm moving from start to goal	The **planned path** preview after *Plan*

Display	Default	Shows
Grid	on	10 cm cells, 2 m × 2 m, in the fixed frame `world`
MotionPlanning	on	Current state, goal state, interactive marker, planned path
RobotModel (Robot PC state)	off	The model from `/robot_description`, posed by the Robot PC's `/tf`
TF	off	All frames of the robot

Camera: left mouse button drag rotates the view around the focal point, middle button (or Shift + left) pans, the wheel zooms. With the **Interact** tool, a drag that starts **on** the marker moves the marker; a drag that starts anywhere else rotates the camera.

3.3 2. Choose a named goal

In the **Planning** tab, check *Planning Group* `arm` and *Start State* `<current>`, then open **Goal State**. The list has the special entries and the named poses from the SRDF:

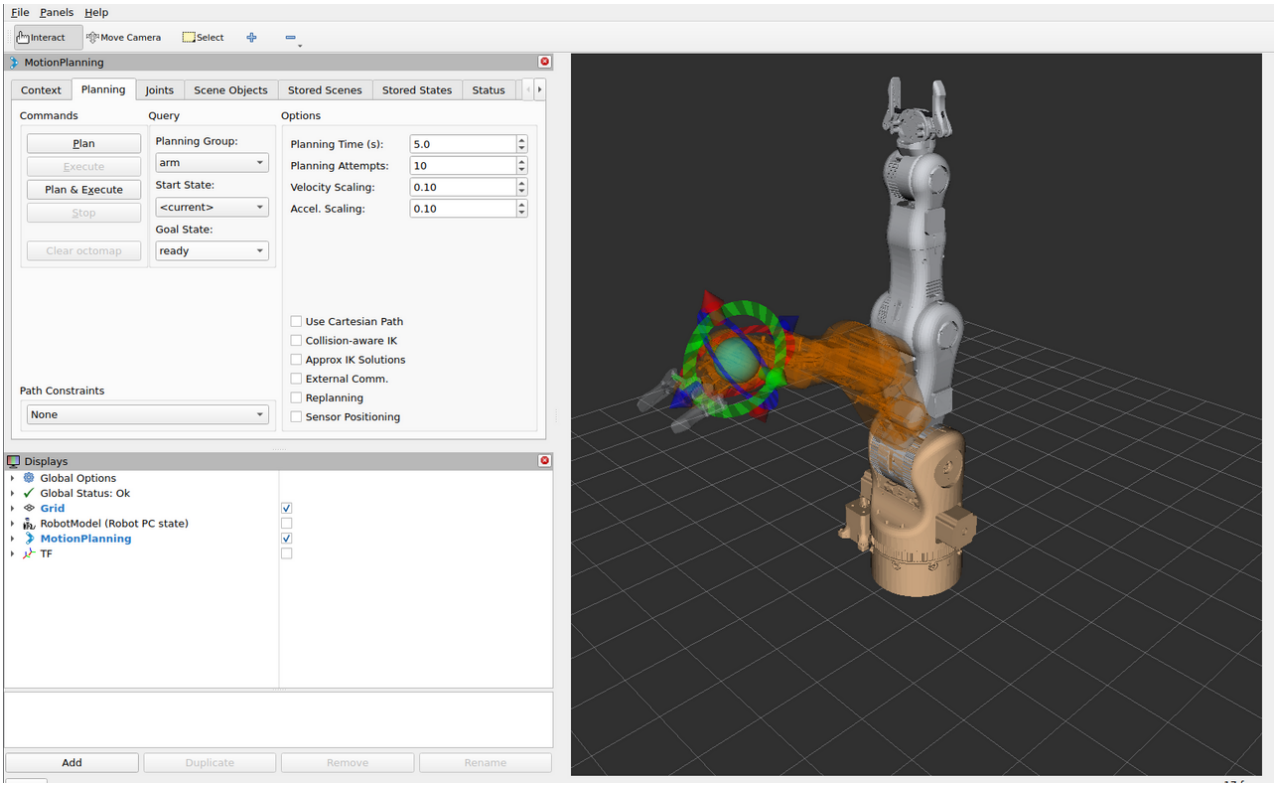


The Goal State list of the arm group: <random valid>, <random>, <current>, <same as start>, <previous>, then the named poses home, ready and cad_pose

Entry	Goal
`<random valid>`	A random collision-free state — useful to exercise the planner
`<random>`	A random state, may be in collision

Entry	Goal
`<current>`	The present state of the arm (no motion)
`<same as start>`	The start state
`<previous>`	The goal before the last change
`home`	All arm joints at 0
`ready`	j2 0.5, j3 0.8, j5 0.6 rad, others 0
`cad_pose`	j2 0.7416, j3 0.4674, j4 0.1426, j5 0.9151 rad — the pose of the BCN3D CAD assembly

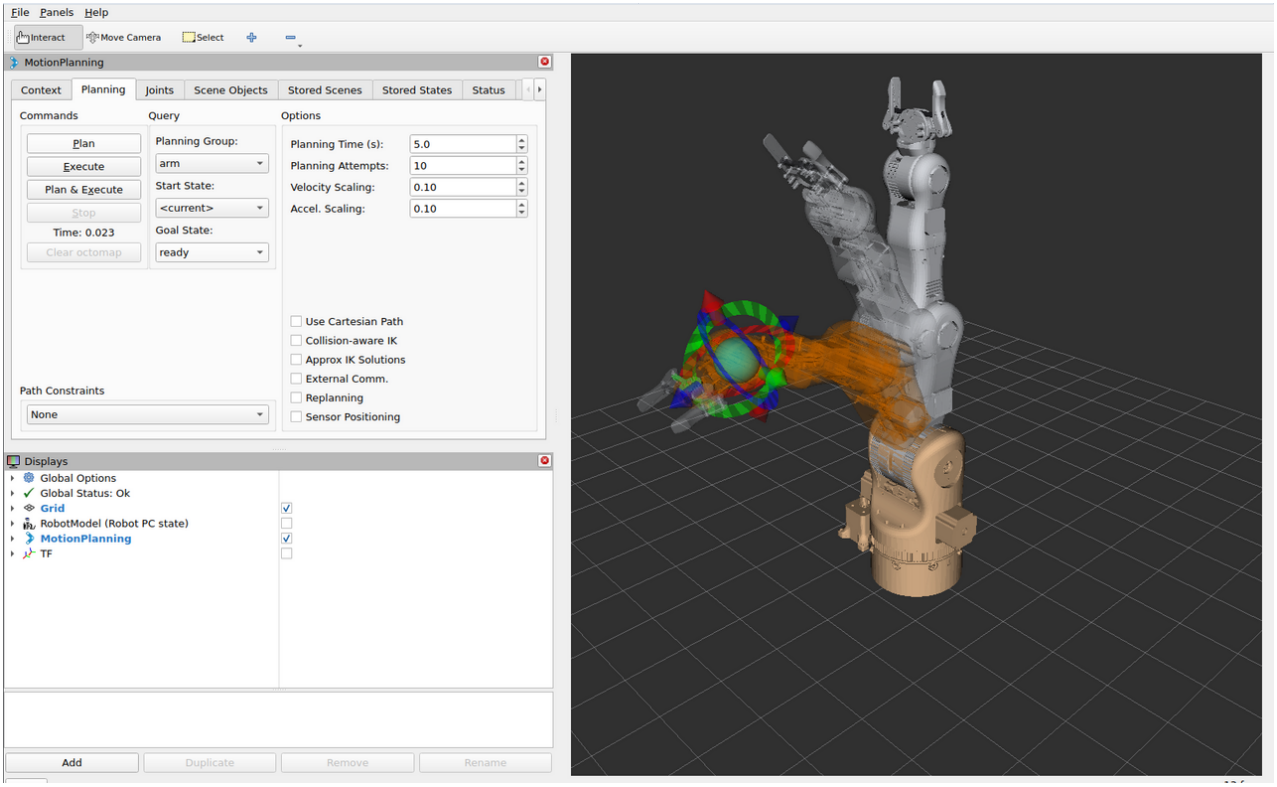
Choose **ready**. The orange goal arm moves to the pose, the current arm stays where it is, and the marker jumps to the goal's `tool_link`:



Goal State ready selected: the orange goal arm bends forward, the grey current arm still stands upright

3.4 3. Plan and check the preview

Click **Plan**. `move_group` plans with OMPL; the time appears under the buttons (here *Time: 0.023* s) and **Execute** becomes active. The grey arm plays the trajectory from the current state to the goal, so you can check the path before anything moves:

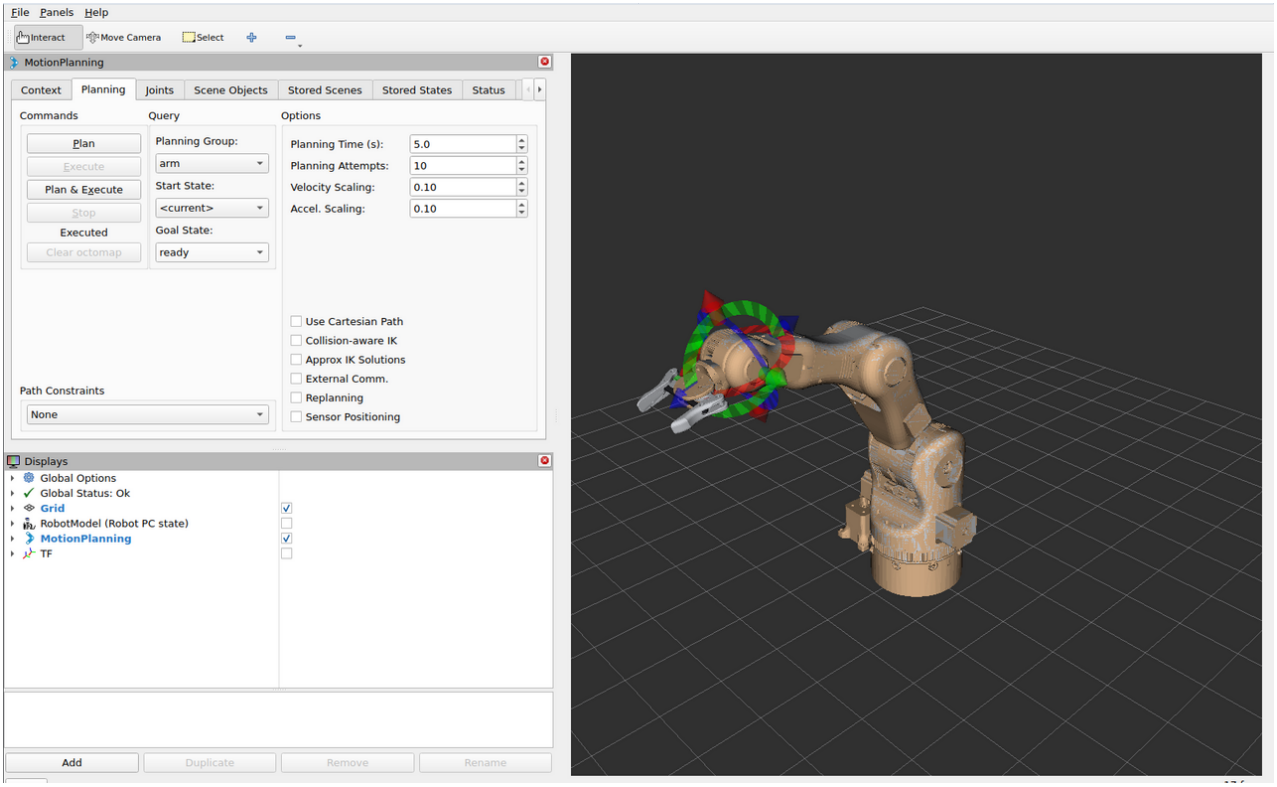


After Plan: Time 0.023 under the buttons, Execute enabled, the grey preview arm moving from the upright start towards the orange goal

If the result is not what you want, change the goal and click **Plan** again; nothing has been sent to the Robot PC yet.

3.5 4. Execute

Click **Execute**. The trajectory goes as a `FollowJointTrajectory` goal to the Robot PC; the status under the buttons changes to *Executed* when the controller reports success, and the current arm stands in the goal pose:



After Execute: status Executed, the arm in the ready pose, current and goal state overlap

The two logs show the same execution from both sides:

```
Operator PC (move_group)
  sending trajectory to arm_controller
  arm_controller started execution
  Goal request accepted!
  Controller 'arm_controller' successfully finished
  Completed trajectory execution with status SUCCEEDED ...

Robot PC (ros2_control_node)
  [arm_controller]: Received new action goal
  [arm_controller]: Accepted new action goal
  [arm_controller]: Goal reached, success!
```

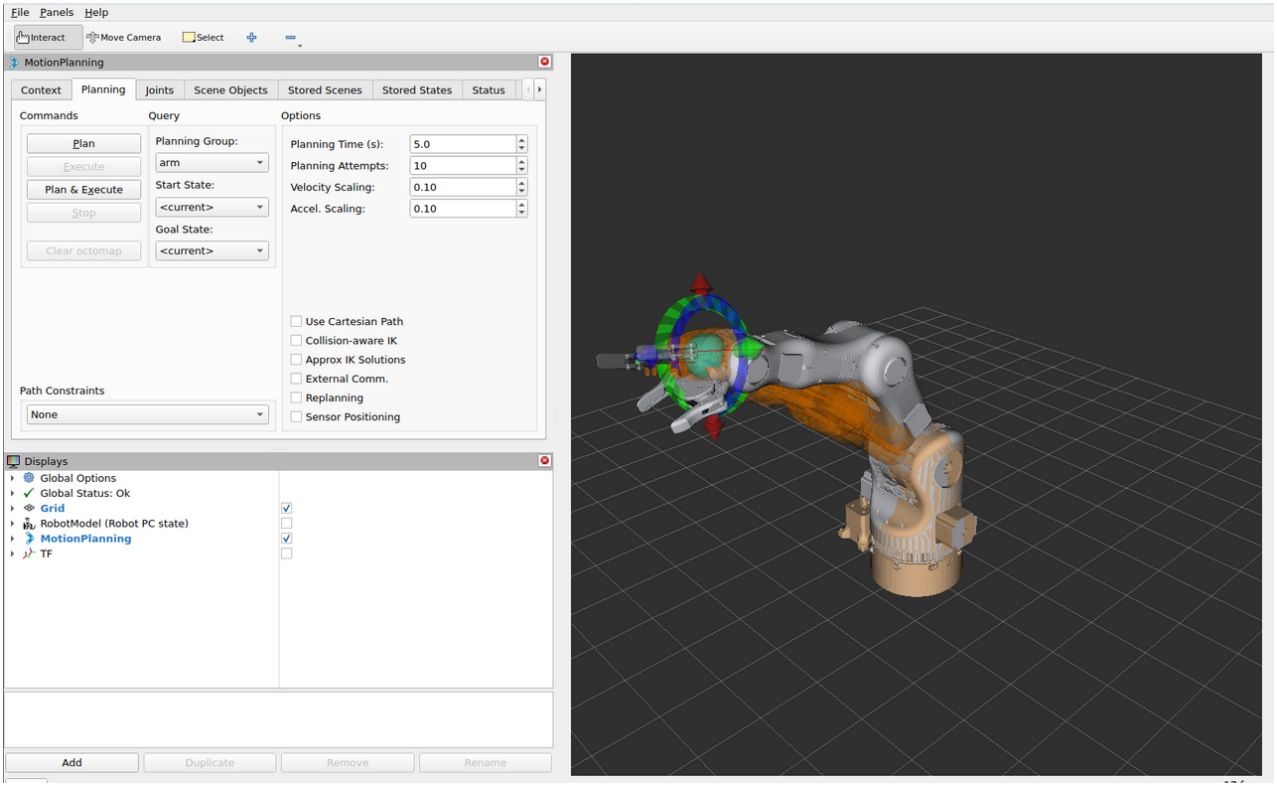
Plan & Execute does steps 3 and 4 in one click, without the chance to check the preview. **Stop** cancels a running execution.

3.6 5. Set a goal with the interactive marker

The marker sits on `tool_link` of the goal state. With the **Interact** tool:

Handle	Moves the goal
Arrow (red, green, blue)	Along that axis of the marker
Ring	Rotation about that axis
Sphere in the centre	Freely in the view plane

Grab a handle and drag. While pick_ik finds a solution, the orange goal arm follows the marker:



The marker dragged along its red arrow: the orange goal arm reaches further forward than the grey current arm

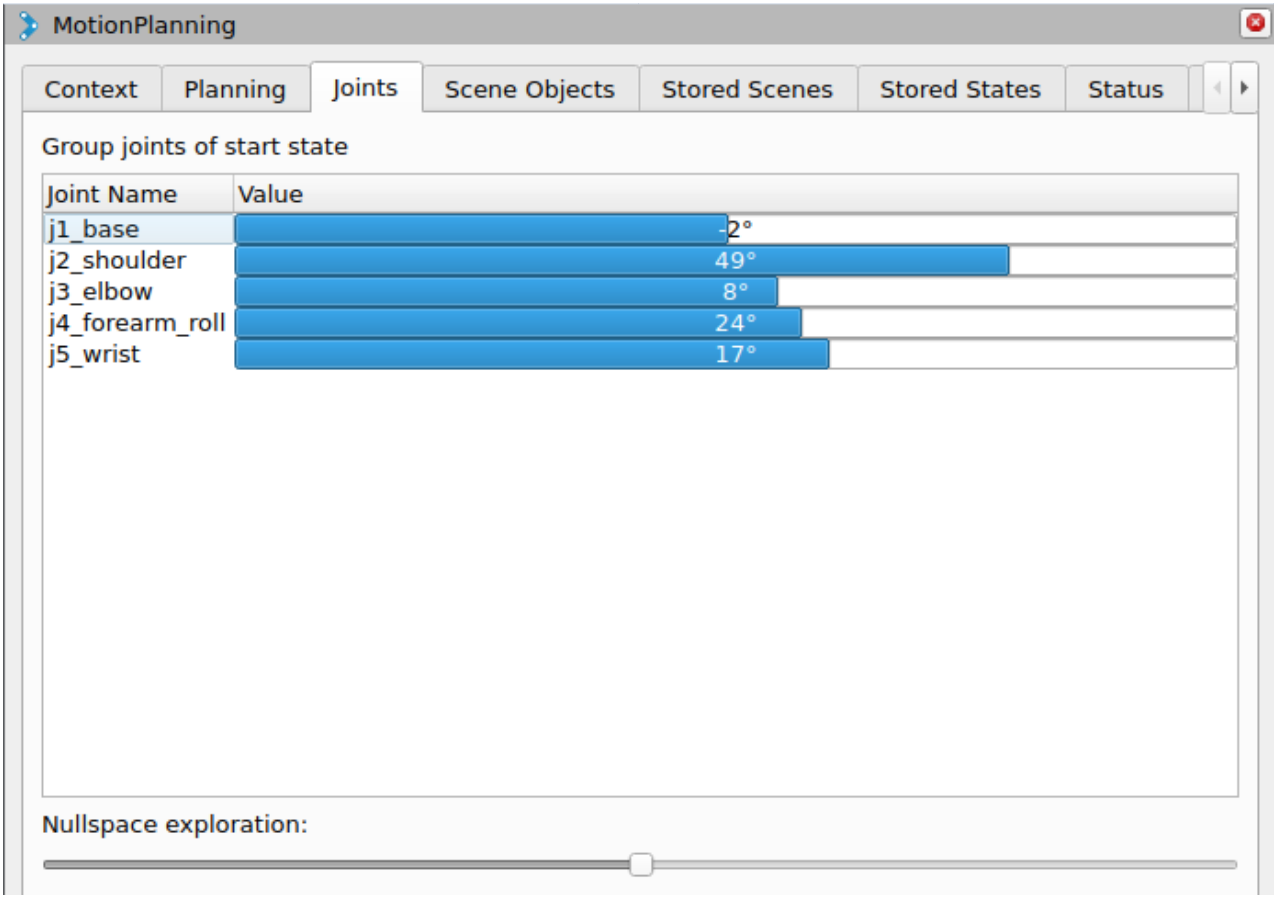
Then **Plan** (and check the preview) and **Execute**, or **Plan & Execute**.

> **Position goals only**
>
> The Moveo has five arm joints, so it cannot reach an arbitrary position *and* orientation. The IK solver matches the tool **position** and ignores its orientation (`rotation_scale: 0.0` in `config/kinematics.yaml`), so the marker always solves for reachable positions; turning the rings has little or no effect on the goal. See MoveIt 2 configuration (p. 32) for a best-effort orientation.

If the marker is dragged out of reach, the goal arm stops following (no IK solution); drag it back towards the base.

3.7 6. Joints tab

The **Joints** tab lists the joints of the planning group with their values in degrees. Drag a bar to change one joint directly — useful for small corrections, for example of `j4_forearm_roll`, which the position-only marker does not set:

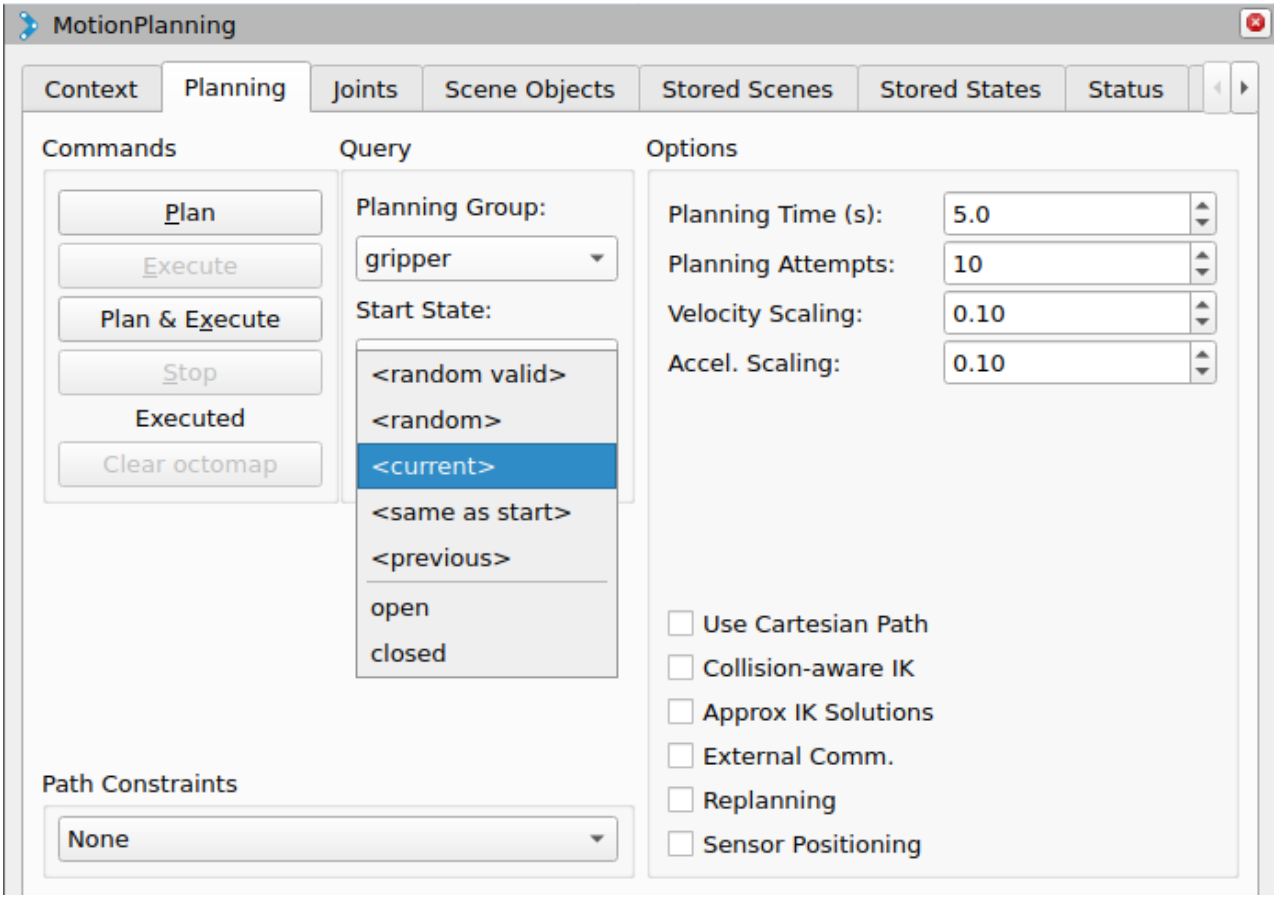


The Joints tab: j1_base -2°, j2_shoulder 49°, j3_elbow 8°, j4_forearm_roll 24°, j5_wrist 17°, with the Nullspace exploration slider below

The tab header names the state the bars belong to. *Nullspace exploration* moves the arm inside the set of joint values that keep the tool where it is.

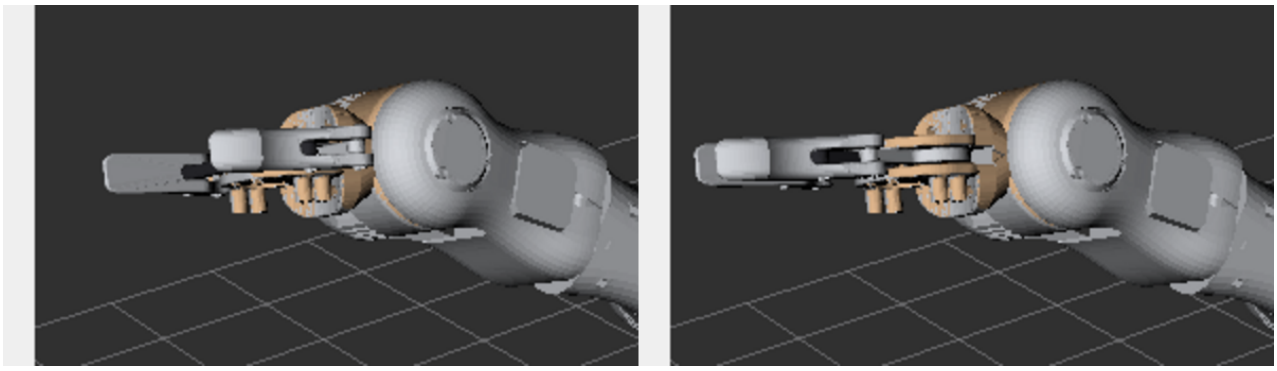
3.8 7. The gripper

In the **Planning** tab, set *Planning Group* to `gripper`. The marker disappears: the gripper is a single joint, not a chain. *Goal State* lists `open` (gripper 0.0 rad) and `closed` (gripper 1.4 rad; limit 1.45):



Planning Group gripper and its Goal State list with the named poses open and closed

Choose **closed** and click **Plan & Execute**. MoveIt sends a one-joint trajectory to `gripper_controller` on the Robot PC; the gears, pivot arms and fingers mimic the `gripper` joint and move with it:



The gripper before (open, left) and after (closed, right) the execution

Switch back to *Planning Group* `arm` for arm motions.

3.9 8. Speed of the motion

The **Velocity Scaling** and **Accel. Scaling** fields in the Planning tab scale the joint limits for each request (0–1). RViz2 starts with **0.10**: the arm moves at 10 % of the limits of the robot description (1.0 rad/s for every joint) and of `config/joint_limits.yaml` (1.0 rad/s²).

Motion (measured with mock hardware, scaling 0.10)	Duration
Arm `home` → `ready`	about 9 s
Gripper `open` → `closed` (1.4 rad)	about 15 s

Raise the scaling for faster motions in simulation. The `default_*_scaling_factor` of 0.3 in `joint_limits.yaml` applies to requests that do not set a scaling, for example from scripts; RViz2 always sends its own value.

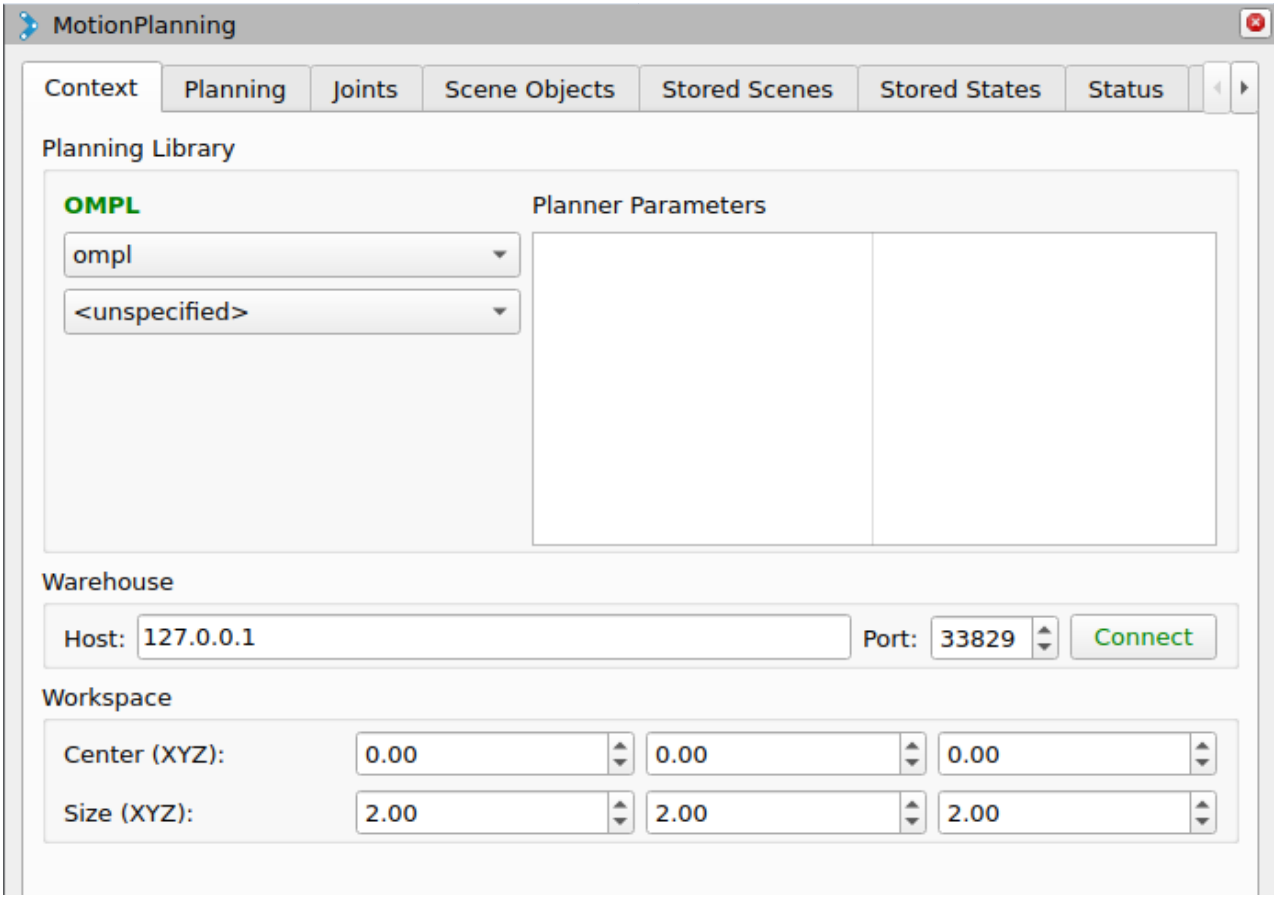
> **Placeholder limits**
>
> The joint limits are placeholders until the Moveo axes are measured on the real arm. With mock hardware every speed works; keep the scaling low when the Robot PC drives real motors.

3.109. Planner options

Planning tab, *Options*:

Option	Default	Meaning
Planning Time (s)	5.0	Time limit per planning attempt
Planning Attempts	10	Attempts; the best result is kept
Velocity / Accel. Scaling	0.10	See above
Use Cartesian Path	off	Straight line of the tool instead of a joint-space path; it needs an IK solution along the whole line, which the 5-DOF arm may not have — leave off unless you need a straight line
Collision-aware IK	off	IK rejects solutions in collision while you drag the marker
Approx IK Solutions	off	Accept approximate IK solutions for the marker
Replanning	off	Plan again if the scene changes during execution

Context tab: the planning pipeline is `ompl`, the only one configured; the planner `<unspecified>` uses the pipeline's default planner. *Warehouse* (a database of stored scenes and states) is not set up and is not needed. *Workspace* limits the planning volume to a 2 m cube around the origin:



The Context tab: planning library OMPL, pipeline ompl, planner unspecified, warehouse host and port, workspace centre 0 0 0 and size 2 2 2

3.11 What happens on Execute

1. RViz2 asks `move_group` to execute the plan.
2. `MoveItSimpleControllerManager` sends it as a `FollowJointTrajectory` goal to `arm_controller/follow_joint_trajectory` (or `gripper_controller/follow_joint_trajectory`) on the Robot PC.
3. The Robot PC's `joint_trajectory_controller` interpolates it at 100 Hz and commands the hardware (mock now).
4. `joint_state_broadcaster` publishes `/joint_states`; `move_group` updates the current state and RViz2 redraws the arm.

MoveIt accepts the execution only if the arm starts within 0.02 rad of the plan's start (`allowed_start_tolerance`), and it aborts it if it takes longer than 1.5 × the planned duration + 1 s. Both settings are in `config/moveit_controllers.yaml` (MoveIt 2 configuration (p. 33)). If an execution fails, see Troubleshooting (p. 43).

3.12 Saving the layout

Panel sizes, the camera and display settings can be saved with **File → Save Config**. The file in the container is part of the image, so a saved change is lost when the container is recreated; to keep it, copy the changes into `ros2_ws/src/smartmanipulator_x_moveit_config/rviz/moveit.rviz` and rebuild.

3.13 move_group without RViz2

To start only `move_group` (for example to plan from a script or from a second RViz2):

```
docker compose -f docker/docker-compose.yml run --rm operator_pc \
    ros2 launch smartmanipulator_x_moveit_config operator.launch.py rviz:=false
```

Stop the `operator_pc` service first: both use the fixed address 10.77.0.20. The `rviz` launch argument (default `true`) is the only argument of `operator.launch.py`.

CHAPTER 4

Development

This page describes how the Operator PC repository is organized and built: the Docker image, the ROS 2 package, the VS Code tasks, the documentation and CI. The Robot PC has its own development page (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/development/).

4.1 Repository layout

```
.
├── docker/
│   ├── Dockerfile                    Operator PC image (ros:jazzy-ros-base + MoveIt 2 +
RViz2 + workspace)
│   ├── docker-compose.yml            operator_pc and shell services on smx_net
│   ├── entrypoint.sh                 sources ROS 2 and the built workspace
│   └── env.example                   template for docker/.env
├── ros2_ws/src/
│   └── smartmanipulator_x_moveit_config/   SRDF, kinematics, limits, controllers, launch,
RViz2
├── tools/docs/                       PDF manual builder, figures, MkDocs hook
├── docs/, overrides/, mkdocs.yml     this documentation
├── requirements-docs.txt             documentation tools (.venv)
└── .vscode/tasks.json                Operator PC and documentation tasks
```

Generated directories (`ros2_ws/build/`, `ros2_ws/install/`, `ros2_ws/log/`, `site/`, `build/`, `docs/downloads/`, `.venv/`) are ignored by Git, and the documentation is kept out of the Docker build context by `.dockerignore`.

4.2 Working on the MoveIt configuration

The workspace is built into the image, not bind-mounted. After a change under `ros2_ws/src/`:

```
docker compose -f docker/docker-compose.yml up --build operator_pc
```

Rules that keep the two computers consistent:

- Do not add a URDF/Xacro here. Change the robot description in the Robot PC repository and rebuild both images.
- Joint names, planning groups, controller names and the joints of each controller must match `smartmanipulator_x_description` and the Robot PC's `controllers.yaml`.
- Keep Operator-only dependencies (MoveIt, RViz2, Gazebo, GUI libraries) in this image; never add them to the Robot PC image.
- The Operator PC gets no USB device and no `ros2_control` hardware interface.

For a quick test of a configuration file without a rebuild, mount it over the installed copy, for example:

```
docker compose -f docker/docker-compose.yml run --rm \
    -v "$PWD/ros2_ws/src/smartmanipulator_x_moveit_config/config:/ws/install/
```

```
smartmanipulator_x_moveit_config/share/smartmanipulator_x_moveit_config/config" \
    operator_pc
```

Stop `operator_pc` first (fixed address); rebuild before you commit.

4.3 VS Code tasks

`.vscode/tasks.json` runs Docker Compose from the repository root. **Ctrl+Shift+B** opens a picker with every task of the build group; none is the default, so the picker always appears.

Task	Group	Command
Operator PC: Start (move_group + RViz2)	build	`docker compose -f docker/docker-compose.yml up --build operator_pc`
Operator PC: Stop	build	`docker compose -f docker/docker-compose.yml down`
Operator PC: Rebuild and restart	build	Stop, then `up --build --force-recreate operator_pc`
Operator PC: Build image	build	`build operator_pc`
Operator PC: Rebuild image (no cache)	build	`build --no-cache --pull operator_pc`
Docs: Build PDF manual	build	`python tools/docs/build_manual.py` in `.venv`
Operator PC: Shell	–	`run --rm shell`
Docs: Live preview	–	`mkdocs serve` in `.venv`

4.4 Documentation

The documentation is the Markdown in `docs/`. MkDocs Material turns it into the website, and `tools/docs/build_manual.py` turns the same pages into the printable A4 user manual (cover, imprint, contents with page numbers, numbered chapters, running headers, PDF bookmarks), rendered with WeasyPrint. Both follow the SystemFromScratch® documentation style of the Robot PC documentation; the landing page is configured in `mkdocs.yml → extra.landing` and the manual in `mkdocs.yml → extra.manual`.

Set up the tools once in the repository `.venv`. WeasyPrint needs the Pango library: on Windows from MSYS2 (`pacman -S mingw-w64-x86_64-pango`; the builder finds `C:\msys64\mingw64\bin` by itself), on Ubuntu `apt install libpango-1.0-0 libpangoft2-1.0-0`.

```
# Windows (PowerShell, repository root)
py -m venv .venv
.\.venv\Scripts\python.exe -m pip install -r requirements-docs.txt
```

```
# Linux (repository root)
python3 -m venv .venv
.venv/bin/python -m pip install -r requirements-docs.txt
```

Command	Result
`python -m mkdocs serve`	Live preview at http://127.0.0.1:8000
`python -m mkdocs build`	Static website in `site/` (strict: a broken link fails the build)
`python tools/docs/build_manual.py`	`build/manual/SmartManipulator-X_Operator_PC_User_Manual_v<version>.pdf`; the version is the latest `vX.Y.Z` tag
`python tools/docs/build_manual.py --output docs/downloads/SmartManipulator-X_Operator_PC_User_Manual.pdf`	The file the website links to (CI does this before deploying)
`python tools/docs/diagrams.py`	Regenerates the hero `docs/assets/images/hero-3d.svg` and the diagrams `system-overview.svg`, `ros-graph.svg`, `control-paths.svg`, `smx-net.svg`, `image-build.svg`
`python tools/docs/diagrams.py --no-hero`	Only the diagrams (no Robot PC checkout needed)

The hero draws the real arm: `tools/docs/hero_robot.py` reads the URDF/Xacro and the visual meshes of `smartmanipulator_x_description` from the Robot PC repository (`SMX_ROBOT_PC_REPO`, default `../SmartManipulator-X-Robot-PC`) and places them by forward kinematics; the glowing tool path above the arm is the fingertip position computed the same way. Run it again after the description changes.

Writing rules that keep the website and the PDF correct:

- Start every page with exactly one `# Title`; chapter and section numbers are added in the PDF, so do not number headings by hand.
- Add a new page to `nav` in `mkdocs.yml`, otherwise it is missing from the site and the PDF.
- An image on a line of its own, `![Caption](assets/images/file.svg)`, becomes a figure with its alt text as the caption (`tools/docs/hooks.py`).
- Link to other pages with relative `.md` paths; strict mode checks them, and the PDF turns them into page references.
- Diagrams are SVG files generated by `tools/docs/diagrams.py`; the PDF renderer runs no JavaScript.
- Every value in the documentation (addresses, limits, parameters) comes from the files of this repository or the Robot PC repository; change the page together with the file.

4.5 CI

Workflow	Trigger	Does
`documentation.yml`	Push to `main`; manual	Builds the PDF manual into `docs/downloads/`, then `mkdocs gh-deploy` publishes the website with the PDF to GitHub Pages (branch `gh-pages`)

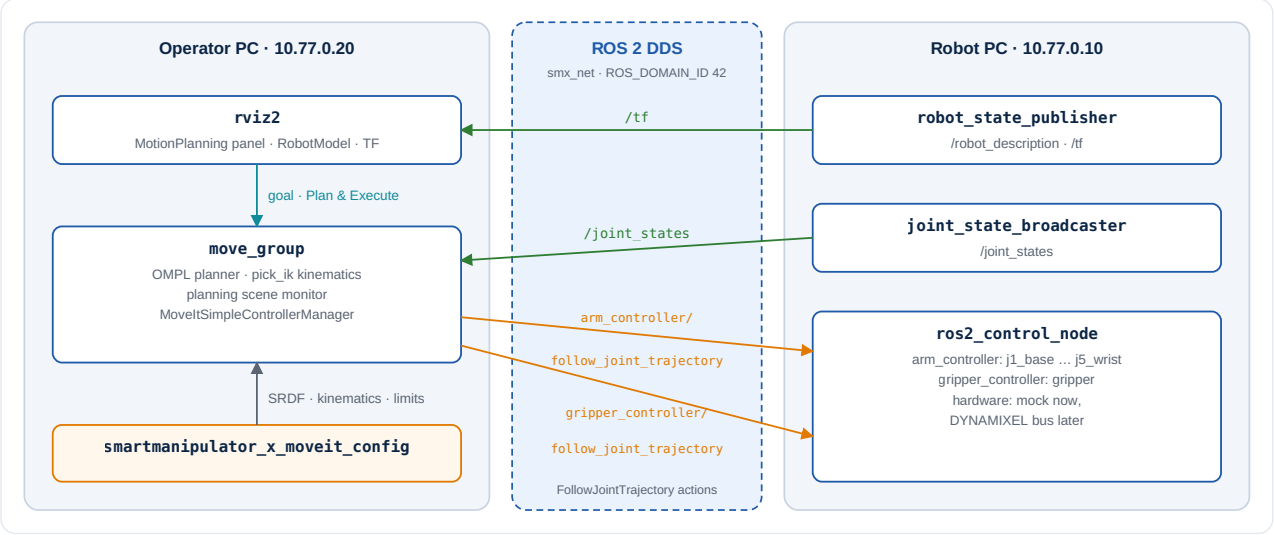
Enable GitHub Pages for the `gh-pages` branch in the repository settings once; the site is then at https://systemfromscratch.github.io/sfs-smartmanipulator-x-operator/.

CHAPTER 5

Architecture

The Operator PC is one of the two ROS 2 computers of SmartManipulator-X. This page shows how its nodes connect to the Robot PC, what crosses the network, and which rules keep the two sides consistent. The architecture of the whole system (controller firmware, FPGA, hardware) is on the system site (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/architecture/).

5.1 ROS 2 graph



ROS 2 graph: rviz2 and move_group on the Operator PC, robot_state_publisher, joint_state_broadcaster and ros2_control_node on the Robot PC, connected over ROS 2 DDS

Node	Com-puter	Started by	Role
`move_group`	Operat-or PC	`operator.launch.py`	Planning (OMPL), IK (pick_ik), planning scene, trajectory execution
`rviz2`	Operat-or PC	`operator.launch.py` (`rviz:=true`)	MotionPlanning plugin, operator inter-face
`robot_state_publisher`	Robot PC	`robot.launch.py`	`/robot_description`, `/tf` from `/joint_states`
`ros2_control_node`	Robot PC	`robot.launch.py`	Controller manager at 100 Hz, hardware interface (mock now)
`joint_state_broadcaster`	Robot PC	spawner	`/joint_states` from the hardware state interfaces
`arm_controller`, `gripper_controller`	Robot PC	spawner	`joint_trajectory_controller`, position command interface

5.2 What crosses the network

Interface	Type	Direction	Used for
`/joint_states`	topic `sensor_msgs/JointState`	Robot PC → Operator PC	Current state of the arm for `move_group` and RViz2
`/tf`, `/tf_static`	topics	Robot PC → Operator PC	Frames of the robot (RViz2 TF and RobotModel displays)
`/robot_description`	topic, transient local	both	Published by `robot_state_publisher` and, with the same model, by `move_group` (`publish_robot_description`); read by the RobotModel display (off by default)
`/arm_controller/follow_joint_trajectory`	action `control_msgs/FollowJointTrajectory`	Operator PC → Robot PC	Arm trajectories (J1–J5)
`/gripper_controller/follow_joint_trajectory`	action `control_msgs/FollowJointTrajectory`	Operator PC → Robot PC	Gripper trajectories
`/display_planned_path`, `/monitored_planning_scene`, MoveIt actions and services		inside the Operator PC	RViz2 ↔ `move_group`

Nothing else is needed from the Robot PC: no controller-manager services, no USB device, no file system.

5.3 The robot model on both sides

`move_group` and RViz2 do not read `/robot_description` from the Robot PC to plan. The launch file expands the **same Xacro file** the Robot PC uses — `smartmanipulator_x_description/urdf/smartmanipulator_x.urdf.xacro` — which the image build copied from the Robot PC repository (see build contexts (p. 36)):

```
MoveItConfigsBuilder("smartmanipulator_x",
package_name="smartmanipulator_x_moveit_config")
    # Same model as the Robot PC; the hardware block is irrelevant to MoveIt.
    .robot_description(file_path=description_xacro, mappings={"use_mock_hardware": "true"}
)
```

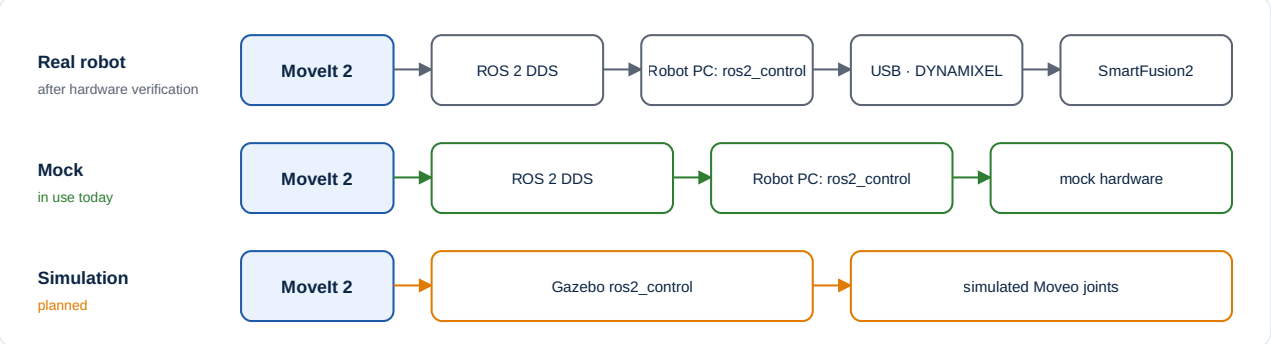
The `use_mock_hardware` mapping only selects the `ros2_control` block of the Xacro, which MoveIt ignores. Links, joints, limits and meshes are identical to the Robot PC's — as long as both images were built from the same Robot PC commit. After a change to the robot description, rebuild **both** images.

5.4 Responsibilities and boundaries

Rule	Why
The Operator PC never opens the controller's USB device; do not pass `/dev/ttyACM0` to it	Only the Robot PC owns the hardware path and its safety
No URDF/Xacro in this repository; consume `smartmanipulator_x_description`	One canonical model, generated from the Moveo CAD
No OpenMANIPULATOR-X model or configuration as a stand-in for the Moveo	Different geometry, joints and actuators
Controller names and joints in `moveit_controllers.yaml` match `controllers.yaml` of the Robot PC	Otherwise MoveIt cannot find the action servers or the joints do not match
MoveIt 2, RViz2 and (later) Gazebo stay out of the Robot PC image	The Robot PC stays lean and headless
ROS 2 data goes over DDS, never over SSH	SSH is for administration only

5.5 Real robot and simulation

The same MoveIt configuration is meant to drive either path; the robot model, joint names, limits, planning groups and controller semantics must agree in both:



One MoveIt configuration, three control paths: the real robot through the Robot PC and the DYNAMIXEL path (after hardware verification), mock hardware on the Robot PC (in use today), and Gazebo simulation (planned)

Today the Operator PC drives the **mock** path. The real path becomes available when the Robot PC enables its verified DYNAMIXEL hardware configuration; nothing changes on the Operator PC for that. Gazebo integration will live in this repository.

5.6 Package layout

```
ros2_ws/src/smartmanipulator_x_moveit_config/
├── package.xml                    ament_cmake; exec-depends on MoveIt, pick_ik, RViz2,
the description
├── CMakeLists.txt                 installs config/, launch/, rviz/
```

```
├── config/
│   ├── smartmanipulator_x.srdf     planning groups, named states, end effector, collision
matrix
│   ├── kinematics.yaml             pick_ik, position-only
│   ├── joint_limits.yaml           accelerations and default scaling
│   └── moveit_controllers.yaml     arm_controller, gripper_controller, execution
tolerances
├── launch/operator.launch.py       move_group + RViz2
└── rviz/moveit.rviz                RViz2 layout
```

Each file is described in MoveIt 2 configuration (p. 31).

CHAPTER 6

MoveIt 2 configuration

`smartmanipulator_x_moveit_config` is the only ROS 2 package of the Operator PC. It holds the semantic description, the kinematics, the planning limits, the controller mapping, the launch file and the RViz2 layout. The robot description itself (URDF/Xacro, meshes) comes from the Robot PC repository.

The configuration is loaded with `MoveItConfigsBuilder` in `launch/operator.launch.py`:

Builder call	File
`robot_description`	`smartmanipulator_x_description/urdf/smartmanipulator_x.urdf.xacro` (Robot PC)
`robot_description_semantic`	`config/smartmanipulator_x.srdf`
`robot_description_kinematics`	`config/kinematics.yaml`
`joint_limits`	`config/joint_limits.yaml`
`trajectory_execution`	`config/moveit_controllers.yaml`
`planning_pipelines`	`ompl` only, default `ompl` (MoveIt's default OMPL settings)
`planning_scene_monitor`	publishes the robot description, the semantic description, the planning scene, geometry, state and transform updates

6.1 Semantic description (SRDF)

`config/smartmanipulator_x.srdf`. Link and joint names come from `smartmanipulator_x_description`.

Planning groups

Group	Definition	Joints
`arm`	chain `base_link` → `tool_link`	`j1_base`, `j2_shoulder`, `j3_elbow`, `j4_forearm_roll`, `j5_wrist`
`gripper`	joint `gripper`	`gripper` (the other gripper joints mimic it)

The end effector `gripper` is attached to `link5`, with `arm` as parent group.

Named states

Group	State	j1_base	j2_shoulder	j3_elbow	j4_forearm_roll	j5_wrist
`arm`	`home`	0	0	0	0	0
`arm`	`ready`	0	0.5	0.8	0	0.6
`arm`	`cad_pose`	0	0.7416	0.4674	0.1426	0.9151

Group	State	gripper
gripper	open	0.0
gripper	closed	1.4

Values in rad. cad_pose is the pose of the BCN3D CAD assembly from which the robot description was generated.

Collision matrix

The SRDF disables collision checking only where contact is part of the design:

- **Adjacent links:** base_link – link1, link1 – link2, link2 – link3, link3 – link4, link4 – link5, and link5 against every gripper part.
- **Gripper parts against each other** (servo_gear, idler_gear, pivot_left, pivot_right, finger_left, finger_right): gears, pivot arms and fingers touch by design.

All other pairs are checked. The convex-hull collision meshes of the description overlap nowhere else at the zero, ready and CAD assembly poses (checked with tools/cad/collide.py in the Robot PC repository). Re-check after the description changes, and add a pair only with its reason.

6.2 Kinematics

config/kinematics.yaml — pick_ik for the arm group:

Parameter	Value	Meaning
kinematics_solver	pick_ik/PickIkPlugin	
kinematics_solver_timeout	0.05 s	Per IK call
kinematics_solver_attempts	3	
mode	global	Global search (finds solutions far from the seed)
position_scale	1.0	The tool position counts fully
rotation_scale	**0.0**	The tool orientation is ignored
position_threshold	0.001 m	A solution must be within 1 mm
orientation_threshold	0.01 rad	Not relevant with rotation_scale 0
cost_threshold	0.001	
minimal_displacement_weight	0.0	No preference for small joint motions
gd_step_size	0.0001	Gradient descent step

Why position-only: the Moveo has five arm joints, so it cannot reach an arbitrary 6-D pose. With a full pose goal, most marker positions in RViz2 would have no solution. With rotation_scale: 0.0 the marker always solves for the tool position. For a best-effort orientation, raise rotation_scale (for example to 0.5): pick_ik then trades position and orientation error, and the result may miss the position by more than 1 mm.

6.3 Joint limits

The position and velocity limits come from the robot description (all arm joints and the gripper: 1.0 rad/s). `config/joint_limits.yaml` adds what the URDF cannot express:

Setting	Value
`default_velocity_scaling_factor`	0.3
`default_acceleration_scaling_factor`	0.3
`max_velocity` (every joint)	1.0 rad/s
`max_acceleration` (every joint)	1.0 rad/s²

The default scaling factors apply to requests that do not set their own scaling, for example from scripts. The RViz2 MotionPlanning panel always sends its own *Velocity Scaling* and *Accel. Scaling*, which start at 0.10 (see Planning in RViz2 (p. 20)).

Joint	Position limits in the description (rad)
`j1_base`	−π … π
`j2_shoulder`	−π/2 … π/2
`j3_elbow`	−π/2 … π/2
`j4_forearm_roll`	−π … π
`j5_wrist`	−π/2 … π/2
`gripper`	0 … 1.45

> **Placeholders**
>
> All limits are placeholders until the Moveo axes are measured on the real arm. Update the position and velocity limits in the robot description (Robot PC repository) and the accelerations here, then rebuild both images.

6.4 Controllers

`config/moveit_controllers.yaml` maps the planning groups to the action servers of the Robot PC. The names must match `controllers.yaml` of the Robot PC repository.

MoveIt controller	Type	Action	Joints
`arm_controller`	`FollowJointTrajectory`	`arm_controller/follow_joint_trajectory`	`j1_base` … `j5_wrist`
`gripper_controller`	`FollowJointTrajectory`	`gripper_controller/follow_joint_trajectory`	`gripper`

Both are `default: true`. The controller manager is `MoveItSimpleControllerManager`: it only needs the action servers, no access to the Robot PC's controller manager.

Execution setting	Value	Effect
`allowed_execution_duration_scaling`	1.5	Execution may take 1.5 × the planned duration…
`allowed_goal_duration_margin`	1.0 s	…plus 1 s, then MoveIt cancels it
`allowed_start_tolerance`	0.02 rad	The arm must start within this of the plan's first point

On the Robot PC side, `arm_controller` and `gripper_controller` are `joint_trajectory_controller/JointTrajectoryController` with a `position` command interface and `position` + `velocity` state interfaces, at 100 Hz.

6.5 Launch file

`launch/operator.launch.py` starts two nodes:

Node	Parameters	Condition
`move_group` (`moveit_ros_move_group`)	the complete MoveIt configuration	always
`rviz2`	`-d rviz/moveit.rviz`; robot description, semantic description, kinematics, planning pipelines, joint limits	`rviz:=true` (default)

```
ros2 launch smartmanipulator_x_moveit_config operator.launch.py            # move_group + RViz2
ros2 launch smartmanipulator_x_moveit_config operator.launch.py rviz:=false  # move_group only
```

6.6 Changing the configuration

1. Edit the files under `ros2_ws/src/smartmanipulator_x_moveit_config/`.
2. Rebuild and restart: `docker compose -f docker/docker-compose.yml up --build operator_pc` (the package is built into the image).
3. If joint names, groups or controllers change, change the Robot PC (`controllers.yaml`, the description) in the same step, and rebuild both images.

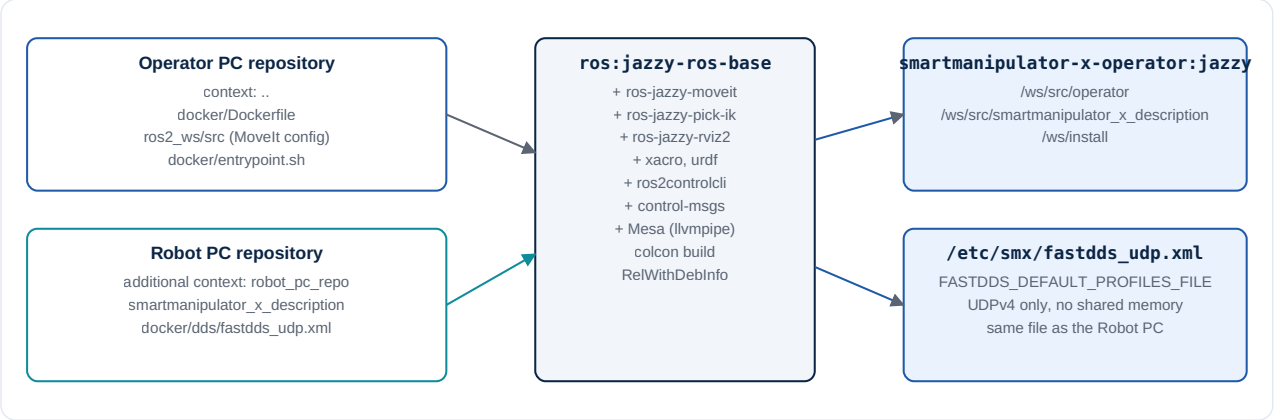
CHAPTER 7

Docker image and Compose

The Operator PC runs entirely in Docker: one image, two Compose services, and the GUI shown on the host through WSLg (Windows 11) or X11 (Linux). ROS 2 is not installed on the host.

File	Content
`docker/Dockerfile`	`ros:jazzy-ros-base` with MoveIt 2, pick_ik, RViz2; builds the workspace
`docker/docker-compose.yml`	`operator_pc` (move_group + RViz2) and `shell` (profile `tools`) on the network `smx_net`
`docker/entrypoint.sh`	Sources ROS 2 and the built workspace, then runs the command
`docker/env.example`	Template for `docker/.env`
`.dockerignore`	Keeps build output, editor files and the documentation out of the build context (the repository root)

7.1 The image



Build contexts of the image: the Operator PC repository and the Robot PC repository feed the ros:jazzy-ros-base build, which produces the smartmanipulator-x-operator:jazzy image with the Fast DDS profile

Step	Content
Base	`ros:jazzy-ros-base` (Ubuntu 24.04, ROS 2 Jazzy)
Packages	`ros-jazzy-moveit`, `ros-jazzy-pick-ik`, `ros-jazzy-rviz2`, `ros-jazzy-xacro`, `ros-jazzy-urdf`, `ros-jazzy-ros2controlcli`, `ros-jazzy-control-msgs`, `python3-colcon-common-extensions`, Mesa (`libgl1-mesa-dri`, `mesa-utils`)
Robot description	`smartmanipulator_x_description` from the Robot PC repository → `/ws/src/smartmanipulator_x_description`
DDS profile	`docker/dds/fastdds_udp.xml` from the Robot PC repository → `/etc/smx/fastdds_udp.xml`

Step	Content
Project packages	`ros2_ws/src` of this repository → `/ws/src/operator`
Build	`colcon build --symlink-install`, `RelWithDebInfo`
Shell	`.bashrc` sources `/opt/ros/jazzy` and `/ws/install`, so `docker exec … bash` has ROS 2 ready

Environment of the image:

Variable	Value
`ROS_DISTRO`	`jazzy`
`ROS_WS`	`/ws`
`RMW_IMPLEMENTATION`	`rmw_fastrtps_cpp`
`ROS_DOMAIN_ID`	`42` (Compose overrides it from `docker/.env`)
`FASTDDS_DEFAULT_PROFILES_FILE`, `FASTRTPS_DEFAULT_PROFILES_FILE`	`/etc/smx/fastdds_udp.xml`
`TZ`	`Europe/Warsaw`

The image tag is `smartmanipulator-x-operator:jazzy` (`SMX_OPERATOR_IMAGE`).

7.2 Build contexts

The robot description and the DDS profile are **not** copied into this repository. The Dockerfile reads them from a second build context, `robot_pc_repo`, which Compose points at the Robot PC checkout:

```yaml
build:
  context: ..
  dockerfile: docker/Dockerfile
  additional_contexts:
    # Canonical robot description and DDS profile.
    robot_pc_repo: ${SMX_ROBOT_PC_REPO:-../../SmartManipulator-X-Robot-PC}
```

```
COPY --from=robot_pc_repo ros2_ws/src/smartmanipulator_x/smartmanipulator_x_description \
     ${ROS_WS}/src/smartmanipulator_x_description
COPY --from=robot_pc_repo docker/dds/fastdds_udp.xml /etc/smx/fastdds_udp.xml
```

So both computers always use the same joints and transport settings. The relative path is resolved from `docker/`; the default expects both repositories side by side. This needs Docker Compose v2.17 or later and BuildKit.

To build without Compose:

```
docker build -f docker/Dockerfile \
    --build-context robot_pc_repo=../SmartManipulator-X-Robot-PC \
    -t smartmanipulator-x-operator:jazzy .
```

7.3 Services

Service	Profile	Network	Command	Use
`operator_pc`	default	`smx_net`, fixed 10.77.0.20 (`SMX_OPERATOR_IP`)	`ros2 launch smartmanipulator_x_moveit_config operator.launch.py`	`move_group` + RViz2
`shell`	`tools`	`smx_net`, dynamic address	`bash` (interactive)	ROS 2 CLI, `rviz2` alone, `glxgears`

Both use `init: true` (a small init process reaps child processes) and the same ROS and GUI environment. `operator_pc` stops with `SIGINT`, like Ctrl+C in a terminal, so `ros2 launch` shuts its nodes down cleanly.

```
docker compose -f docker/docker-compose.yml up --build operator_pc     # start
docker compose -f docker/docker-compose.yml run --rm shell             # shell
docker compose -f docker/docker-compose.yml build operator_pc          # rebuild only
docker compose -f docker/docker-compose.yml build --no-cache --pull operator_pc   # from
scratch
docker compose -f docker/docker-compose.yml down                       # remove containers
```

`smx_net` is declared `external`: it belongs to the Robot PC Compose project, which creates it. `down` here leaves it in place.

7.4 Environment

Every value has a default in `docker-compose.yml`. Copy `docker/env.example` to `docker/.env` only to change one; `docker/.env` is ignored by Git.

Variable	Default	Meaning
`ROS_DOMAIN_ID`	`42`	Must match the Robot PC
`SMX_OPERATOR_IP`	`10.77.0.20`	Address of `operator_pc` on `smx_net` (the Robot PC is 10.77.0.10)
`SMX_ROBOT_PC_REPO`	`../../SmartManipulator-X-Robot-PC`	Robot PC checkout, relative to `docker/` or absolute
`SMX_OPERATOR_IMAGE`	`smartmanipulator-x-operator:jazzy`	Image name and tag
`SMX_X11_DIR`	`/run/desktop/mnt/host/wslg/.X11-unix`	Host directory mounted as `/tmp/.X11-unix`
`SMX_WSLG_DIR`	`/run/desktop/mnt/host/wslg`	Host directory mounted as `/mnt/wslg`
`SMX_DISPLAY`	`:0`	`DISPLAY`
`SMX_WAYLAND_DISPLAY`	`wayland-0`	`WAYLAND_DISPLAY`
`SMX_XDG_RUNTIME_DIR`	`/mnt/wslg/runtime-dir`	`XDG_RUNTIME_DIR`
`SMX_PULSE_SERVER`	`/mnt/wslg/PulseServer`	`PULSE_SERVER` (WSLg audio; unused by RViz2)

Variable	Default	Meaning
`SMX_LIBGL_ALWAYS_SOFTWARE`	`1`	`1`: software OpenGL (llvmpipe)

Fixed in the Compose file: `RMW_IMPLEMENTATION=rmw_fastrtps_cpp`, `ROS_AUTOMATIC_DISCOVERY_RANGE=SUBNET`, `QT_QPA_PLATFORM=xcb`, `QT_X11_NO_MITSHM=1`.

7.5 GUI on Windows 11 (WSLg)

The defaults are made for Docker Desktop with the WSL 2 backend. Docker Desktop exposes the WSLg sockets of its VM under `/run/desktop/mnt/host/wslg`; Compose mounts them into the container, and RViz2 draws through X11 (`QT_QPA_PLATFORM=xcb`) on the WSLg display `:0`. Nothing has to be configured; run the Compose commands from PowerShell or from a WSL distribution.

7.6 GUI on a Linux desktop (X11)

Allow local containers to connect to your X server once per session, and set the X11 values in `docker/.env`:

```
xhost +local:docker
cp docker/env.example docker/.env
```

```
# docker/.env
SMX_X11_DIR=/tmp/.X11-unix
SMX_WSLG_DIR=/tmp/smx-no-wslg
SMX_DISPLAY=:0
SMX_XDG_RUNTIME_DIR=/tmp
```

`SMX_DISPLAY` must match `echo $DISPLAY` on the host. `SMX_WSLG_DIR` only needs to be a path that exists or can be created; nothing is used from it on Linux.

7.7 Rendering

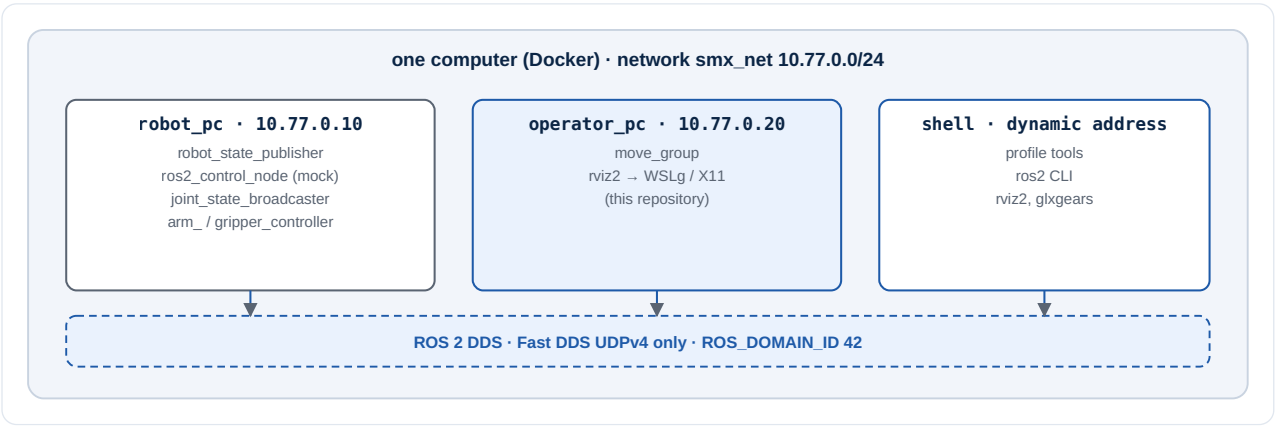
`LIBGL_ALWAYS_SOFTWARE=1` (Mesa llvmpipe) is the default because it works on every host, without GPU drivers in the container. RViz2 with the Moveo model is usable this way; it costs CPU. With a working GPU path (for example `/dev/dri` passed through on Linux with a Compose override, or the WSLg D3D12 driver), set `SMX_LIBGL_ALWAYS_SOFTWARE=0`. Check with `glxinfo -B` or `glxgears` in the `shell` service which renderer is in use.

CHAPTER 8

Network and DDS

The Operator PC and the Robot PC talk only through ROS 2 DDS, like two computers on an Ethernet. On one computer, the Docker network `smx_net` stands in for that cable; on two computers, the real network does. The domain ID and the DDS profile stay the same in both cases.

8.1 One computer: smx_net



The Docker network smx_net on one computer: robot_pc at 10.77.0.10, operator_pc at 10.77.0.20 and the shell container, all on ROS 2 DDS with Fast DDS over UDPv4 and ROS_DOMAIN_ID 42

Setting	Robot PC	Operator PC
Network	creates `smx_net` (10.77.0.0/24, `SMX_SUBNET`)	joins it as `external`
Address	10.77.0.10 (`SMX_ROBOT_IP`)	10.77.0.20 (`SMX_OPERATOR_IP`)
`ROS_DOMAIN_ID`	42	42
`RMW_IMPLEMENTATION`	`rmw_fastrtps_cpp`	`rmw_fastrtps_cpp`
Fast DDS profile	`docker/dds/fastdds_udp.xml`	the same file, copied at build time to `/etc/smx/fastdds_udp.xml`
`ROS_AUTOMATIC_DISCOVERY_RANGE`		`SUBNET`

8.2 The Fast DDS profile: UDP only

Fast DDS uses shared memory between participants it believes are on the same host. Containers on one computer have separate `/dev/shm`, so discovery can succeed while no data arrives — the nodes are listed, but topics stay silent. The profile disables the built-in transports and keeps all traffic on UDPv4:

```
<participant profile_name="smx_udp_only" is_default_profile="true">
  <rtps>
    <userTransports>
```

```
      <transport_id>smx_udpv4</transport_id>
    </userTransports>
    <useBuiltinTransports>false</useBuiltinTransports>
  </rtps>
</participant>
```

UDP is also what the real Ethernet link carries, so simulation and hardware use the same transport. The file belongs to the Robot PC repository; the image build copies it, so a change there reaches the Operator PC with the next `--build`.

8.3 Two computers: real Ethernet

When the Robot PC moves to its own board (Raspberry Pi CM4, Jetson or an Ubuntu 24.04 PC), replace `smx_net` by the host network on **both** sides:

1. In both Compose files, remove the `networks` entries of the services and the `smx_net` declaration, and add `network_mode: host`.
2. Keep `ROS_DOMAIN_ID`, `RMW_IMPLEMENTATION` and the Fast DDS profile unchanged.
3. Make sure both computers are in one subnet and the firewall lets DDS through (UDP, ports 7400 and up for domain 42 — the RTPS ports are computed from the domain ID; the simplest rule allows UDP between the two hosts).
4. Check discovery from both sides with `ros2 node list`, then `ros2 topic echo /joint_states --once` on the Operator PC.

`network_mode: host` gives a container the network of the Docker host. On Linux that is the real network. With Docker Desktop on Windows the Docker host is a VM, so a host-mode container is not necessarily reachable from the Robot PC; verify discovery in both directions, or run the Operator PC on a Linux computer for two-host operation.

8.4 Native Windows ROS 2 cannot join

ROS 2 installed natively on Windows does not discover the containers on `smx_net`: Docker Desktop does not route multicast between Windows and its container networks. Run every ROS 2 node of the system in a container on that network — for the command line, use the `shell` service.

8.5 SSH is not a ROS transport

SSH is for administration (logs, updates, starting containers on the Robot PC). ROS 2 topics and actions always go over DDS; do not tunnel or forward them through SSH.

8.6 Checklist

Check	Command (in the `shell` service)	Expected
Same domain	`echo $ROS_DOMAIN_ID`	`42` on both sides
Discovery	`ros2 node list`	`/move_group`, `/rviz2`, `/robot_state_publisher`, `/controller_manager`, …

Check	Command (in the `shell` service)	Expected
Joint states	`ros2 topic echo /joint_states --once`	Six joint names with positions
Control-lers	`ros2 control list_controllers`	`arm_controller`, `gripper_controller`, `joint_state_broadcaster` active
Actions	`ros2 action list`	`/arm_controller/follow_joint_trajectory`, `/gripper_controller/follow_joint_trajectory`
Profile	`echo $FASTDDS_DEFAULT_PROFILES_FILE`	`/etc/smx/fastdds_udp.xml`

CHAPTER 9

Troubleshooting

Most problems fall into four groups: the containers cannot start, the GUI does not show, the two computers do not see each other, or MoveIt plans but the Robot PC does not execute. Start every diagnosis with a shell on the network:

```
docker compose -f docker/docker-compose.yml run --rm shell
```

9.1 Normal messages

Two `ERROR` lines appear at every start and are harmless:

Message	Meaning
`No 3D sensor plugin(s) defined for octomap updates` (`move_group`)	No depth camera is configured
`Action server: /recognize_objects not available` (`rviz2`)	No object recognition behind the *Detect* button of the Scene Objects tab

9.2 Start and build

Symptom	Cause	Fix
`network smx_net declared as external, but could not be found`	The Robot PC is not running, so `smx_net` does not exist	Start the Robot PC first (`up robot_pc` in its repository)
`additional_contexts` is rejected, or `unknown flag: --build-context`	Docker Compose older than v2.17, or BuildKit disabled	Update Docker Desktop / the Compose plugin; do not set `DOCKER_BUILDKIT=0`
`failed to compute cache key: "/ros2_ws/src/smartmanipulator_x/smartmanipulator_x_description": not found`	The Robot PC repository is not at `../SmartManipulator-X-Robot-PC`	Clone it side by side, or set `SMX_ROBOT_PC_REPO` in `docker/.env`
`Address already in use` for 10.77.0.20	A second `operator_pc` container (for example from `run`) holds the fixed address	`docker compose -f docker/docker-compose.yml down`, then start once
`/entrypoint.sh: no such file or directory`	The script was checked out with CRLF line endings	The Dockerfile strips CR already; re-clone with `.gitattributes` in effect (LF) and rebuild

9.3 Display

Symptom	Cause	Fix
`could not connect to display :0` / `qt.qpa.xcb: could not connect`	The X11 socket is not mounted, or the wrong `DISPLAY`	Windows: Docker Desktop with WSL 2 backend and WSLg (Windows 11). Linux: set the X11 values in `docker/.env` and run `xhost +local:docker`
`Authorization required, but no authorization protocol specified`	The X server refuses the container	Linux: `xhost +local:docker` in the desktop session
Window opens, but the 3D view is black or RViz2 crashes on start	OpenGL driver problem	Keep `SMX_LIBGL_ALWAYS_SOFTWARE=1` (default); test with `glxgears` in the `shell` service
RViz2 is slow	Software rendering on few cores	Expected with llvmpipe; set up a GPU path and `SMX_LIBGL_ALWAYS_SOFTWARE=0`, or close other displays

9.4 Discovery and data

Symptom	Cause	Fix
`ros2 node list` shows only the local nodes	Different `ROS_DOMAIN_ID`, or not on the same network	Same domain on both sides (default 42); both containers on `smx_net` (`docker network inspect smx_net`)
Nodes are listed, but `ros2 topic echo /joint_states` stays silent	Fast DDS chose shared memory between containers	Check `FASTDDS_DEFAULT_PROFILES_FILE=/etc/smx/fastdds_udp.xml` on both sides; rebuild the image if the file is missing
Native Windows ROS 2 does not see the containers	Docker Desktop does not route multicast between Windows and container networks	Use the `shell` service instead
Discovery works on one computer, not on two	Firewall or different subnets	Allow UDP between the hosts, same subnet, see Network and DDS

9.5 Planning and execution

Symptom	Cause	Fix
`move_group` waits with `Didn't receive robot state (joint angles) with recent timestamp`	No `/joint_states` from the Robot PC	Check the Robot PC is running and the discovery checks above
The robot in RViz2 does not move after Execute	The Robot PC did not accept the goal	Look at the Robot PC log; `ros2 control list_controllers` must show `arm_controller` active
`Unable to identify any set of controllers that can actuate the specified joints`	Controller names or joints differ between `moveit_controllers.yaml` and the Robot PC's `controllers.yaml`	Make them identical; rebuild

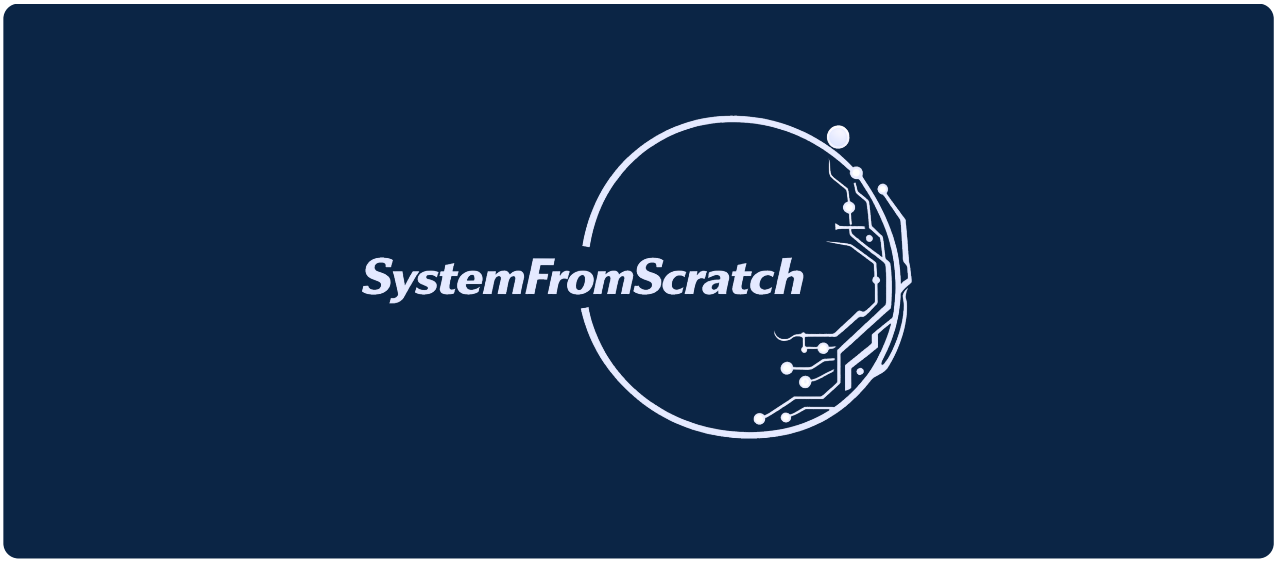
Symptom	Cause	Fix
`Action client not connected to action server: arm_controller/follow_joint_trajectory`	The controller is not running, or DDS does not deliver actions	`ros2 action list` in the `shell`; restart the Robot PC
Execution aborted: start state deviates	The arm is more than 0.02 rad away from the plan's start	Plan again from the current state (*Start State* `<current>`)
Execution aborted after a time	The trajectory took longer than 1.5 × planned + 1 s	Lower the speed scaling, or check the Robot PC controller (100 Hz) is not overloaded
The marker does not follow, goal turns red	No IK solution: the position is out of reach, or the goal is in collision	Move the marker closer to the base; check joint limits
Rotating the marker does nothing	Expected: the IK ignores orientation (`rotation_scale: 0.0`)	See kinematics
`Planning failed` for reachable goals	OMPL found no path in the planning time, or the start state is in collision	*Planning Time* up in the Context tab; *Start State* `<current>`; check the collision matrix

9.6 Model mismatch

If the arm in RViz2 and the Robot PC disagree (different joints, a missing link, errors about unknown joints), the two images were built from different versions of the robot description. Rebuild both from the same Robot PC commit:

```
# in the Robot PC repository
docker compose -f docker/docker-compose.yml up --build robot_pc
# in this repository
docker compose -f docker/docker-compose.yml up --build operator_pc
```

Author & Company



SmartManipulator-X and its Operator PC are designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification.

Author

Anton Vynohradov · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Embedded Systems Engineer · Secure Product Architect · Founder of SystemFromScratch®

Embedded systems built from scratch — from bare metal to Linux, from first schematic to certification.

Anton designs and builds reliable, secure embedded products end to end: firmware, board support packages and drivers, RTOS and Embedded Linux platforms, hardware bring-up, and the security and compliance work needed to ship them. He is based in Poland and works remotely worldwide.

Area	Focus
Embedded software	Bare-metal, RTOS and Linux firmware in C/C++, built to be maintained for years
BSP & drivers	Bootloaders, Zephyr drivers and Linux kernel work for MCUs, SoCs and SoC FPGAs
Security & compliance	Secure boot, threat modelling, MISRA C/C++, EU Cyber Resilience Act (CRA) readiness
Industries	Industrial IoT, automotive, robotics, medical and consumer electronics

Technology: C · C++ · Python · Zephyr RTOS · FreeRTOS · Embedded Linux · Yocto · Buildroot · ARM Cortex-M · STM32 · Microchip FPGA · RISC-V · CAN / ISO-TP · ROS 2 · Docker · Git · CI/CD · MISRA C/C++ · EU CRA

For SmartManipulator-X, Anton designed the system architecture, the Robot PC and Operator PC images and their ROS 2 configuration, the MoveIt 2 configuration of the Moveo, the robot description generated from the BCN3D Moveo CAD, the SmartFusion2 controller firmware with its motor tester, and this documentation.

SystemFromScratch®

SystemFromScratch® was founded by Anton Vynohradov. The company combines deep embedded engineering with cybersecurity and certification-focused product development, for teams in industrial IoT, automotive, robotics, medical and consumer electronics that need a partner who understands the whole stack — silicon, firmware, operating system and the regulations that govern them.

Service	What it covers
Embedded software development	Bare-metal, RTOS and Linux firmware for performance, reliability and long-term maintainability
BSP & device drivers	Board support packages, bootloaders and low-level drivers for MCUs, SoCs, FPGAs and custom hardware
RTOS & Embedded Linux	Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot): porting, configuration, integration, reproducible builds
Hardware bring-up & integration	Schematic review, first boot, validation and production support, including SoC FPGA and motor-control platforms
Cybersecurity & CRA readiness	Secure boot, firmware integrity, threat modelling and a secure development lifecycle for the EU Cyber Resilience Act
Compliance & consulting	MISRA C/C++, static analysis, code and architecture reviews, compliance roadmaps

SmartManipulator-X is one of the open-source projects in the SystemFromScratch® portfolio: public engineering work that can be inspected line by line, from the ROS 2 host down to the stepper driver.

Contact

Channel	Address
Website	systemfromscratch.com
E-mail	avynohradov@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch
LinkedIn	linkedin.com/in/vynohradov
ORCID	0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Channel	Address
Location	Poland — working remotely worldwide

Building a robot, bringing up new hardware, or preparing for the Cyber Resilience Act? Get in touch through systemfromscratch.com or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue on GitHub (github.com/SystemFromScratch/sfs-smartmanipulator-x-operator/issues). Please include the commits of this repository and of the Robot PC repository, the host OS, the Docker and Compose versions, the command you ran and its output (the `move_group` log for planning problems).
- **Contributions:** read Development (p. 24) first; keep the documentation in step with the change.
- **Commercial support, integration and custom development:** contact SystemFromScratch® through systemfromscratch.com.

Trademarks and acknowledgements

SystemFromScratch® is a registered trademark. The SystemFromScratch® name and logo are used in this documentation to identify the publisher of SmartManipulator-X; the Apache License 2.0 grants no rights to use them, and they may not be used to endorse or promote products derived from SmartManipulator-X without prior written permission.

ROS is a trademark of Open Source Robotics Foundation, Inc. MoveIt is a trademark of PickNik Inc. ROBOTIS, DYNAMIXEL and OpenMANIPULATOR are trademarks of ROBOTIS Co., Ltd. SmartFusion2 is a trademark of Microchip Technology Inc. BCN3D is a trademark of BCN3D Technologies. Docker is a trademark of Docker, Inc. Windows is a trademark of Microsoft Corporation. SmartManipulator-X is not affiliated with or endorsed by these companies; their names are used only to identify the products the project builds on.

The mechanical design comes from the BCN3D Moveo (github.com/BCN3D/BCN3D-Moveo) project (MIT license; the license text ships with the robot description meshes). The ROS 2 software concepts come from ROBOTIS OpenMANIPULATOR-X (github.com/ROBOTIS-GIT/open_manipulator) and the DYNAMIXEL SDK and hardware interface. Motion planning uses MoveIt 2 (moveit.ai) with pick_ik (github.com/PickNikRobotics/pick_ik) and OMPL; the operator interface is RViz2.

APPENDIX A

License

The project-owned files of SmartManipulator-X are free and open-source software, licensed under the **Apache License, Version 2.0** (apache.org/licenses/LICENSE-2.0), matching the ROBOTIS Open-MANIPULATOR-X ROS 2 repository.

Item	Value
Copyright	© 2026 Anton Vynohradov
License	Apache License 2.0
SPDX identifier	`Apache-2.0`
Applies to	The Operator PC image files, the ROS 2 package, the tools and this documentation

Third-party sources keep their own licenses: the ROS 2, MoveIt 2 and pick_ik packages installed in the image from the ROS package repository, and the BCN3D Moveo meshes of the robot description that the image build takes from the Robot PC repository (MIT, text in `meshes/LICENSE-BCN3D-Moveo` there).

A.1 What the license allows

In short — the full text below is authoritative:

- **Use** the project in any project, open source or proprietary, commercial or not.
- **Modify** it and **distribute** it, in source or binary form, alone or as part of a product.
- **Patent grant:** each contributor grants you a license to any of their patents that the contribution uses.

A.2 What it requires

When you distribute SmartManipulator-X or a modified version of it:

- Include a copy of the license.
- Keep the copyright, patent, trademark and attribution notices in the source files.
- Mark the files you changed.
- If you redistribute a NOTICE file, keep its attribution notices (SmartManipulator-X has none at the moment).

A.3 What it does not give

- No warranty: the software is provided "as is".
- No trademark rights: the license does not let you use the names of the author or of System-FromScratch® to endorse your product.

A.4 Trademarks

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. The marks of Open Source Robotics Foundation, PickNik, ROBOTIS, Microchip, BCN3D, Docker and Microsoft are listed on the About (p. 45) page.

A.5 Full license text


```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
```

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the

```
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```